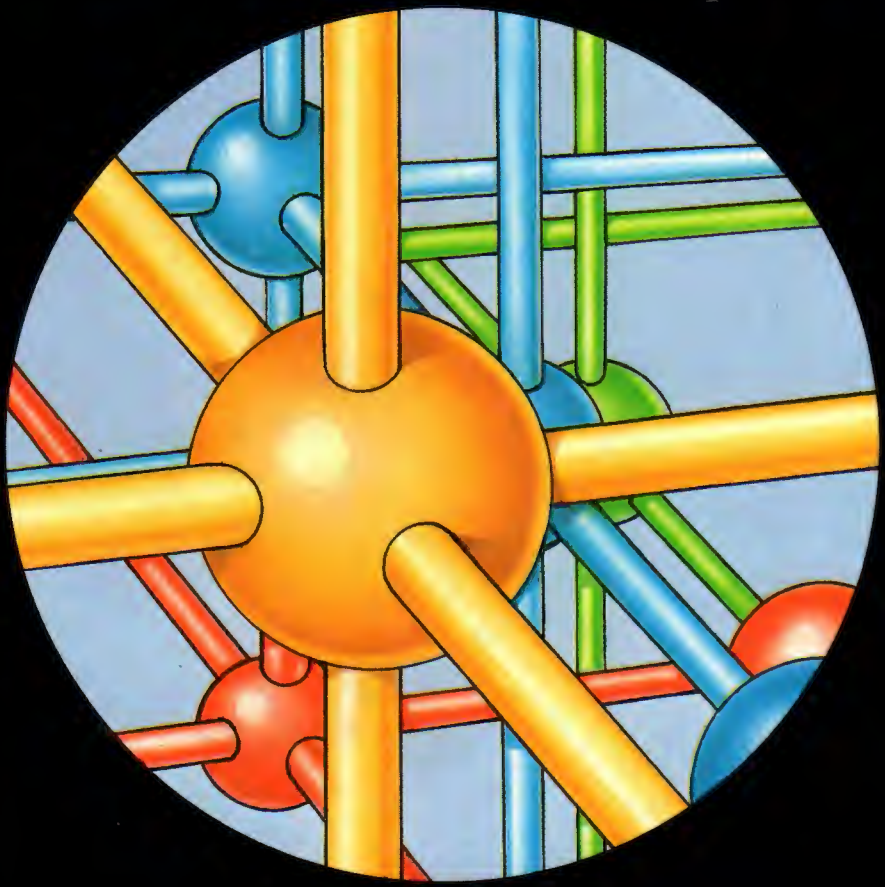
3D CONSTRUCTION KIT 2.0™



INSTRUCTION MANUAL

ENGLISH

incentive
The Award Winners

DOMARK

CONTENTS

REGISTRATION 1
ACKNOWLEDGEMENTS 2

PREFACE 3

TUTORIAL

1 Introduction to 3D 4
2 "Learning to walk and learning to fly again"
3 "Of palaces and bungalows – A house of our own" 7
4 "Beam me up Spotty" 12
5 "Holy Exploding Cubes Batman!!" 18
6 "Faders and Sensors and Making your own Videos" 23
7 "A Player of Games, and a few loose ends" 27

REFERENCE GUIDE

8 The user interface reference guide 37
8.1 Conventions 37
8.2 The file selector 38
8.3 The text editor 40
8.4 The alert box 41
8.5 The dialogue box 41
8.6 Guide to the screen 42
8.6.1 The status bar 42
8.6.2 The freescape controls 44
8.6.3 The view controls 44
8.6.4 The shortcut icons 45
8.6.5 The object editor icons 45
8.6.6 The colour panel 47
8.6.7 The debugger screen 48

9 The menus reference guide 49
9.1 The file menu 49
9.2 The general menu 52
9.3 The area menu 59
9.4 The object menu 60
9.5 The conditions menu 63

10 FCL tutorial 66

11 FCL reference guide 76
11.1 Introduction 76
11.2 Constants 77
11.3 Variables 77
11.4 Animators 80

12 FCL command reference guide 81
12.1 Format of guide 81
12.2 Alphabetical listing of FCL functions 83
13 Sound editors 164
14 Make 169

Appendix 1 170
A1.1 Installation and Loading Instructions 170
A1.2 Filename Extensions 171
A1.3 Loading Objects / Worlds from the Clip-Art Catalogue 171
A1.4 Importing Objects from 3D Construction Kit 1.0 171

Appendix 2 172
Hints and tips 172
Appendix 3 174
Keyboard shortcuts 174

REGISTRATION

It is essential to register as a 3D Construction Kit user, as support can only be given to registered owners. The registration form is included with the package.

All correspondence should be sent to Mandy Rodrigues, at the address shown below. If a reply is required a stamped addressed envelope must be enclosed.

THE 3D CONSTRUCTION KIT USERS' CLUB

The Club is provided to offer additional help and advice for users of the 3D Construction Kit and will consist of a bi-monthly newsletter packed full of news, information, hints and tips on the system to allow everyone to use it to its full potential. It will also act as a forum for users to exchange ideas and information. To apply for membership of the Club, just fill in the relevant section of the registration card and further details and information will be sent to you. All registration forms should be sent to:

Mandy Rodrigues, 67 Lloyd Street, Llandudno, Gwynedd, LL30 2YP.

NOTICE

ACKNOWLEDGEMENTS

Original concept and design:	Ian Andrew
Programming and design:	Paul Gregory (Amiga & ST) Kevin Parker (PC)
Clip art and borders:	The Kremlin Peter D. Ward Robin Ball Mieke V.D. Poll Marc Epstein Liam Johnston Stephen Day Sue Medley Martin Sullivan S. Shield S.M. Hindley Steve Rogers Stefan Goetzke
Special thanks to Michael Adams	
Freescape development:	Chris Andrew
User manual:	Jason Orbaum and Colin Boswell
Sound editors:	Oxford Mobius and Dave Chapman
Video production:	Status Visual Communications
Production:	Colin Boswell and the Domark Software Group

Many thanks to Mandy Rodrigues

Preface

Not only have you purchased a tool for manipulating objects in a virtual three-dimensional world, but also a complex, but easy-to-learn command language which will allow you to bring the world you create to life. Although 3D Construction Kit 2.0 is built on a complex and accurate base of mathematical concepts, it has a user-friendly interface which uses concepts familiar to everyone. By the far the best way to learn any product, is to roll your sleeves up and get stuck in, so we've included a step-by-step tutorial which will get you going. There is also a complete reference guide covering everything you always wanted to know about 3D Construction Kit 2.0 (and weren't afraid to ask!). But most of all remember, 3D Construction Kit 2.0 is about having fun, whether you are creating complex interactive worlds or just colouring a simple scene. The only limit is your imagination.

Welcome to the world of 3D Construction Kit 2.0!

The soap-box bit : much time and energy have gone in to a product like 3D Construction Kit 2.0. Not just in creating the software, but in also providing on-going support for you, the User. We have not copy-protected 3D Construction Kit 2.0, because we feel that you should be able to make copies of the disk for your own personal use. Please do not abuse our trust by giving copies to other people.

1: An Introduction to Three Dimensions

Before we get to the tutorial section of the manual it is important to familiarise ourselves with the concepts and terms we will be using. Don't worry if some of this seems confusing at first – a couple of sessions with the program and the tutorial and you'll be talking jargon with the best of them. A couple of sessions after that and you might even understand what you're talking about. That's the way the human brain works, and it's got you this far so its way too late to worry now.

The younger reader of this weighty but zesty tome will have no trouble with three dimensional geometry (scary words – but their bark's worse than their bite). Those, however, whose memories of geometry involve slates, abacuses, or monotonous teachers in equally monotonous suits, and those who had actually forgotten what the word geometry means will be pleased to hear that all is to be made clear in the next few pages.

If you have an understanding of the concepts of the program, or if you are an experienced user of this program's illustrious predecessor, simply turn the machine on and skip to the section 2 of the tutorial. Those who are itching to start can also do this and read the "why"s later... this is one of the reasons why books are still a medium to be reckoned with.

One, two, and three dimensions

Imagine a line, or better yet draw one. Imagine a creature who exists on that line. The only way the creature can move is along the line. It can go forwards or backwards along the line. If we want to tell someone else where the creature is we only need one number to tell them. Provided we both measure from the same point on the line – the zero point, and we both measure in the same units (centimetres, inches, kilometres – it doesn't matter so long as we are all using the same unit), we need simply say "The creature is 7 units down the line" and we will be talking about the same place. This number of units will change as the creature moves – the number varies from time to time, so we call it a **variable**. For distance on the line, we will call the variable **"X"**. It's not a good name as names go, but James Bond never asked "Q" about his name so we'll simply follow his example here.

The creature we have been describing is moving in one dimension – it is moving on a line and can only go forwards and backwards. Now let us imagine that the creature learns how to move not just forwards and backwards, but to move sideways as well. We now need to say not only how far down the line the creature is but how far to the side of it it is. We need two numbers, another variable. History has named this second variable **"Y"**.

We are now at a stage we can recognise: the variables to describe where a point is in a space are known as **co-ordinates**. Map references are usually two co-ordinates (the "X" and "Y" or "along" and "up" to the more language minded, or even "latitude" and "longitude" for those of a cartography bent). Battleships, one of the eldest games (certainly older than the actual battleships themselves!) is played by players who give each other "X" and "Y" co-ordinates to exchange shots. Even when we refer to "row five, column three" we are expressing two dimensional co-ordinates.

Let us imagine now that the creature develops a third way of moving. It realises that not only can it move forwards, backwards, and sideways, it can also move up and down. We now need a third variable, one that expresses how far above the line the creature is. It will come as no surprise to those that know their alphabet that this variable is named "Z". It is the third co-ordinate that allows us to express where an object is in three dimensional space. Now go back and read that last sentence again.

So, a quick recap/bluffers guide. If we are looking at a horizontal line on a page from above, the X co-ordinate tells us how far along the line to go, the Y co-ordinate tells us how far up the page to go, and the Z co-ordinate tells us how far above the page to go.

That's the principle. The problem is that a computer screen is a two-dimensional image – it doesn't have depth, so we must simulate three dimensions with complex mathematics (the same way that an artist does subconsciously when they draw a three dimensional space on a piece of paper). On the screen our X tells us how far along the object is, our Y is its height from the ground line and our Z is how far away from us the object is. To put it yet another way (confusing the issue even more) the **"X – Axis"** is the left-right line, the **"Y – Axis"** is the up down line, and the **"Z – Axis"** starts between your eyes and ends in the mysterious world "inside" the screen. All axes intersect at a single point: the zero point – co-ordinates X=0, Y=0, Z=0, or (0,0,0) for short. It is conventional for co-ordinates to be put in brackets, this is to confuse students of language who were getting cocky after the X-Y-Z trick. In terms of 3D Construction Kit 2.0, the zero point is at ground level on the "front" left of the screen. Believe me, this will fall into place once you see it in practice.

Worlds, areas, and objects

3D Construction Kit 2.0 allows you to create a **world**. This is a pretty mammoth task. It took God six days and even he/she/it/they (delete with caution) needed a rest afterwards. To make the task easier, a world in 3D Construction Kit 2.0 is made out of a number of linked **areas**. An area might be the inside of a house, or a housing estate, or a weird cosmic plane inhabited by spiky eight-legged estate agents, the only limitation to what the area is is your imagination. Just like in the real world, areas can be completely different (if you don't believe me try a quick America-India-Albania round trip). There is one similarity between these areas however : they all contain objects.

Objects are cubes and spheres and hexagons and lines and all sorts of prettily named shapes – they are the building bricks which we group together to make houses, tables, giant teaspoons, purple haddock, or whatever takes our fancy. Once we have grouped a few objects together to make something we call this something a **group**. Nothing too shocking about that.

Within the confines of 3D Construction Kit 2.0 we are offered many objects which we can bend, stretch, move, colour, twist, turn, and group together pretty much as we like. Remember:- the objects are the components that we build the groups out of; the groups are placed in the areas; the world consists of all the areas. Understanding these terms now will save much head scratching and nail chewing later.

Well, that's it for the introduction – see it wasn't that painful after all. Now there's no time for further stalling, get that program running... this is where the fun starts.

2: Learning to walk again and then learning to fly for the first time

Turn on your machine and run the program. Loading instructions for each version of 3D Construction Kit 2.0 are different but if you are having any trouble you will find them in Appendix One at the back of this manual.

Figure 2.1

The screen should look like fig. 2.1 (only in colour – obviously!). If the screen looks vastly different then consult the loading instructions again (reading them left to right and working down the page this time) and try again. If the program still refuses to load or fails to come up with the control panel as shown (fig. 2.1) then you've probably got a corrupted disk, or a broken computer. The traditional response to this is to cry, swear, hit the machine/disk a few times (but not too hard as it costs a lot of money), and then contact the manufacturer. The first parts of this are not essential but have been shown to be stress relieving. If the screen shows carefully arranged rows and columns of square-edged aliens who seem to be marching left and right and slowly descending towards a blob with a dot on it at the bottom of the screen then you are reading the wrong manual.

This screen is the heart of the 3D Construction Kit 2.0 editing system and consists of three separate and distinct parts. The top line of the screen is known as the **menu bar.** Moving the mouse will move the arrow on the screen and if the right button is clicked on one of the words on this bar a **menu** will appear. A menu, as its name suggests, is a series of options from which one makes a selection. Most computer users are familiar with menus but their use will be explained later for the uninitiated.

The bottom third of the screen is taken up by the **control panel.** This panel is your main interface with the editor. Moving the arrow onto one of the many buttons and clicking will activate the button and produce the desired effect (or, at least, the correct effect). If the control panel looks complex at the moment, don't worry. Once you have used it a few times it will seem easy and logical, as indeed it is.

Between the control panel and the menu bar is the **view.** This area is where it all happens, it is a window onto the world, from the user's point of view, where we can move around and interact with the objects we will insert. At the moment it looks like three coloured bars but it is in fact... the horizon. The blue bar at the top is the sky (do I really need to explain this?), the green bit is the ground, and the grey bit is a **floor.** The floor differs from the sky and the ground because it is an object and can be removed from the area. The sky and ground are always present, but their colours can be changed or made identical (for interiors).

The control panel is also divided into three sections. The first section contains the **information window**. Three lines of text can be seen in the window at a time. To see the next line down you simply point the arrow at the lower of the two buttons on the left of the window and push the left button (this manoeuvre is known as a **point and click** maneouvre for obvious reasons). To move back up the page press the upper button. Little arrows have been thoughtfully placed on the buttons, so there's no need to waste memory cells.

Try these buttons, and you will see the text in the window change from an indecipherable series of seemingly unconnected words and numbers to a different but equally confusing set of the same things. Still, at least we know the buttons work. Take the window back to the top of the page (the first three letters of the top line should read POS) and we'll move on to the next bit of the control panel.

The next section of the control panel are the editing buttons. Each of these buttons enable us to manipulate objects or the area in some way. We will be launching into this in part three of the tutorial but for now we'll use one of them to put an object in the area. For this we need the second button along. The first button is the Incentive Logo, clicking on this will bring up the program credits which can be removed by clicking on OK or by pressing the return key.

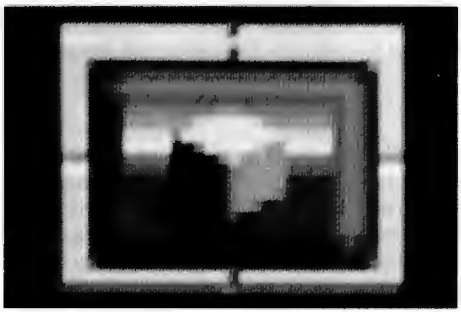


Figure 2.2

Click on the second button along, the one shown in fig.2.2. The box shown in fig. 2.3 will appear. Click on the top left button in this selector (the one with a drawing of a cube on it). The selector will disappear and a cube will appear in view, hovering magically above the ground, to the right of the centre cross, and with its base just below the horizon. Do not panic if you do not understand what you just did. All will be explained in Section Two. For now, we have simply put the cube there so that when we practice moving we have something to move around.

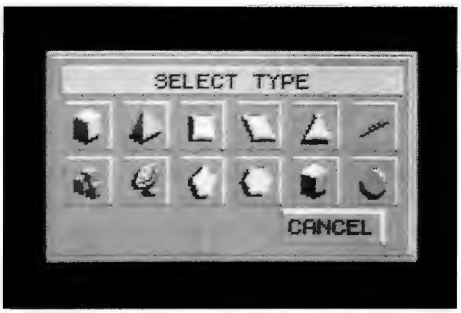


Figure 2.3

The final section of the control panel is the movement controller. The dazzling array of buttons enable the "player" to move around the area on each of the three axes of movement (X, Y and Z), and also to turn their head in all directions. We will now look at all the buttons in this section of the control panel in detail. Try each of them as we describe them, and be sure you understand their use before you move on to the next.

The first button on the top row is a **toggle** button. This means that it turns an item off if it is on, or on if it is off.. like the touch on/off controls on many modern televisions or stereo systems. This first button toggles the centre cross in the view area. The cross is useful for positioning, but can get in the way later when testing.

The button directly underneath it turns you round exactly 180 degrees (a half turn). When you press this button the cube will disappear from the view because it will now be behind you. This button, by the nature of what it does, is also a toggle, so pressing it once more will bring the cube back. When you turn to stand with the cube behind you the second number on the second row of the information window will change from 0 to 180. The three numbers on this row represent your **angle of rotation** on the three axis (X, Y, and Z).

Angles are measured in degrees. There are 360 degrees in a circle (or a full turn), and therefore it follows that a half turn is 180 degrees and a quarter turn is 90 degrees. Degrees are measured clockwise and are an indication of which *direction* you are pointing in.

As has already been stated there are three axis in three dimensional space. Things get confusing because the X axis is used for horizontal measurement but to control vertical direction. Things become simpler with the use of a simple analogy. Imagine that you a ball with one eye skewered through the centre from left to right on a long skewer (okay, so the analogy isn't that simple). We can move the ball left and right on this skewer (the X axis), or by turning the skewer we can make the eye look down or up. Thus, the axis of rotation is the axis on which we turn our view, not the axis upon which our view appears to move.

To save some confusing thinking here is a summary of movement and rotation on the various axes:

X Axis: Move left and right. Look up and down.

Y Axis: Move up and down. Look left and right.

Z Axis: Move forwards and backwards. *Tilt* head clockwise and anticlockwise.

To turn slowly left or right (rotate on the Y axis) we use the second and fourth buttons on the top row. Our Y rotation changes in the information window. We move in steps of five degrees every time we click the left mouse button, and thirty degrees when we press the right mouse button. This is the same on all the rotation buttons, but the numbers can be changed by the user. We will do this later.

The buttons directly below the rotate left and right move the player to the left and right. The first and third numbers on the top row of the information screen will change when

this button is clicked. This row gives the co-ordinates of the location of your viewpoint in the area. You are in walk mode, ie. you cannot fly so can only move on the ground. Your mode of viewing is shown on the third line of the screen. We will be changing it later in this section but for now it is best to get to grips with walking. Once you have fallen to the ground your Y position will not change when you take side steps, but your X position will change by 40 units every time you click with the left button and 200 units every time you click with the right button. These figures apply for movement on all three axis.

The middle two buttons in this set of six move you forwards and backwards in the direction you are facing. In walk mode this is restricted to rotations on the Y axis but when flying things get a little more interesting.

To the right of this main movement control are two buttons with underlined arrows. These move you upwards and downwards (on the Y axis). If you are in contact with the ground and cannot go through it they will cause you to duck and stand up (your height, the last unexplained number in the information window, will change according to its own limits).

To the left of these two buttons are a blue divider and another set of three buttons. The two smaller buttons will look up and down (rotate on the X axis) and the tall thin button will return your gaze to looking straight ahead (X **rot**ation = 0).

The final two buttons on the far right control the tilt of your head (rotation on the Z axis). And that's the lot.

Try moving and looking in all directions and then getting back to your initial position (co-ordinates 4000,310,4000). If things become confusing and you want to get back the quick way click on the far right button on the bottom row. This will reset you to your start position before falling, as though you had just switched the program on. It will not alter any of the objects you have been working on, it simply resets you.

Once you've mastered walking, it's time to fly. Click on **"General"** on the Menu bar. The General Menu will appear. The first Item on this Menu is **"Mode"**. It is used to control our modes of movement and our vision. Click on Mode and a second Menu will appear to the immediate right of the first. We want the top option – vehicle – so click on this. The two menus disappear and yet another one appears (it's a bit like a recurring nightmare isn't it). The line of buttons at the top of the menu control your movement modes and viewpoint. You can make an **absolute** move, that is one which takes you to a specific location specified by co-ordinates, by altering the figures at the bottom half of this menu but for now we are interested in only three buttons.

Walk should already be highlighted. Next to it are **Fly 1** and **Fly 2**. To select one of these two modes of movement simply click on it. In Fly 1 your movement is identical to that in walk mode with the added benefit that you can fly up and down as well. Select it, click on OK, and try it out.

Now change your movement to mode Fly 2. The **path** to this route is *General – Mode – Vehicle – Fly 2*. To follow the path simply click on each of the named buttons in turn. We will be using paths a lot, especially in the reference section, so it's a good idea to get used to following them now.

Try experimenting in Fly 2. See if you can work out what the difference between the two Fly Modes is. If you give up, or think you've got it, return to this very page and read on.

In Fly 1, when you press to go forwards only your X and Z co-ordinates can be changed – i.e movement in the area is always parallel to the ground as it is taken only in terms of your rotation about the Y-axis. In Mode Fly 2 however your rotations about all the axis are taken into account and you move in a straight line along your sightline. This allows totally free flight but tends to end in tears and confusion. To add to the frustration, using the emergency reset (bottom right icon button) also resets your movement to its **default** (initial) state – Walk. There is a way to change this default, but for now a bit of practice with menus won't do any harm.

Once you've got the hang of moving in the area and flying round the cube, things start to get a bit boring... so don't just sit there wandering around an empty space: let's build a world.

3: Of palaces and bungalows – A house of our own

Whilst moving around is great fun and has been a source of constant enjoyment to countless generations of infants, it can be made even more thrilling when there are things to move around – Space Invaders would not have been a big hit if there were no invaders to shoot at.

Turn on your machine and load up 3D Construction Kit 2.0. You will be presented with the menu bar, view, and control panel. The only object in the area is the floor. This is where we will build our houses.

The bottom row of the control panel contains the **icon** buttons. Icons are pictures that indicate the function of the button. The first thing we need to do is put the body of the house down. For this, we will need a cube. The icon to create an object is the cube. It is the first real button on the bottom row (although the incentive button works it has no real value and is not even drawn as a button). When we click on this we are presented with the menu shown in fig. 3.1.

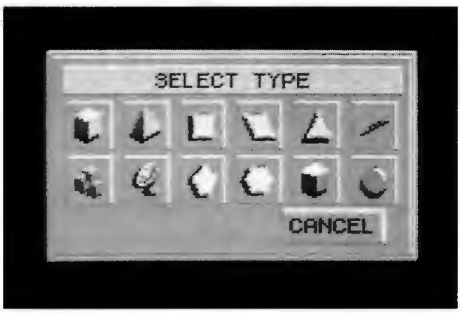


Figure 3.1

This panel has twelve square buttons and a cancel button. You'll have to work out what Cancel does on your own (yes, you've guessed it), but I'll run through what the others do.

The top row of six buttons create objects. What this means is they bring the object into the view and add the name of the object to the bottom of the object list. We can change the name of an object but we cannot change its position on the object list. It is an object's position that gives it its number and when we are telling the machine what an object does we always refer to it by number. We will be using the object list a lot, so it is an important concept to understand. When an object is deleted, the other objects below it on the list do not change position. The space remaining is filled by the next object created.

The objects created by the top row are, from left to right, a **cube** (six sided solid), a **pyramid** (solid with four sides and an openable top), a **square** (two sided four edged plate with fixed 90 degree corners), a **quad** (two sided four edged plate with changeable corner angles), a **triangle** (two sided three edged plate with changeable corner angles), and a **line** (single sided with movable ends, always straight).

A **solid** is a three dimensional shape like a box and a **plate** is a two dimensional (flat) shape like a piece of paper. Sides of a shape are its surfaces (the colourable bits) and **edges** are the straight boundaries around the sides.

The first two buttons on the bottom row are slightly different – the first of them creates a group (more on this later on in the section), and the second creates a sensor (forget about it for now!). The last four buttons create objects as follows: a **pentagon** (two sided five edged plate with changeable corner angles), a **hexagon** (two sided six edged plate with changeable corner angles), a **flexicube** (like a cube but with changeable corner angles – think of it as a cube made up of triangular building bricks), and a sphere (always perfectly round but size is alterable).

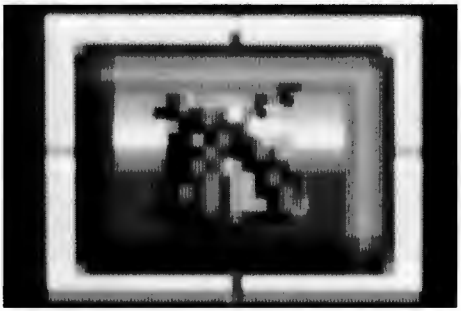


Figure 3.2

If you select the wrong object you will need to delete it. To do this press the second icon button as shown in fig.3.2, this is the **Delete Object** icon. You will be presented with a list and hopefully unsurprised to find out that it is the object list. The first object on the list will be the floor, and underneath it will be listed any objects you have created. The currently selected object will be written underneath the list, and to the right of it will be a button marked Select. There are two ways of selecting an object, either click on its name on the list, or click on select and then click on the object itself in the view window. Once you have selected the right object for deletion, click on OK. You are presented with a warning, click on Yes if you are sure you want to go ahead with the deletion. The object will be removed from both the list and the view and you can carry on.

Try creating and deleting some objects – you won't be able to see the plates if there is something else behind them of the same colour – create them on their own and they are clearly visible. The objects are placed in the top right quarter of the view, so you can move around and then create objects to avoid them lying on top of each other. Once you've got this understood, return to your initial view (bottom right icon) and delete everything except the floor. Then create a cube. It will be object number one on the list. This is to be the body of the house.

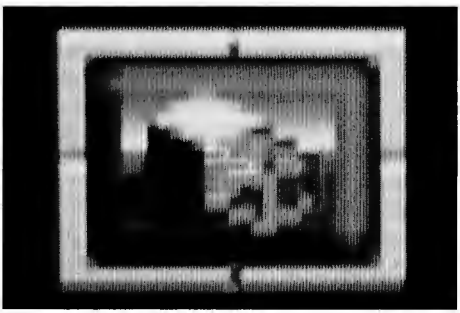


Figure 3.3

Fig. 3.3 shows the icon button for the command **Edit Object.** It is the sixth icon button along. Click on this, select the cube, and then take a deep breath. The bottom half of the control panel changes, the information window shows the fourth, fifth, and sixth lines, and you are now in edit object mode.

The three lines of text tell you the number and name of the object, the co-ordinates of the origin of the object, and the dimensions of the object (length, height, and depth, or to put it another way, X, Y, and Z).

The bottom panel section is divided into six parts. They are divided by blue title lines that give a pretty big hint as to the functions of the buttons in each part. The first part, "point", is shaded out, this tool is not available on the cube and we will use it when we put the roof on the house.

The second part is labelled **Turn** and its function, not too shockingly, is turning the object through 90 degrees on any of the three axis. The object is turned about its centre so its position remains unchanged. Its far easier to work out what the individual functions do by messing about with them on some object so we will explain what each part does in theory and you can get the specifics by trying them out. Basically, each of the buttons controls the function of each part of the panel on each of the axes (there, that made things clearer didn't it!).

Shrink allows you to move the faces (or edges on a plate) of the object towards each other. The centre point will change and thus both object position and size are affected. The next part of the panel, **Stretch**, does exactly the same thing in reverse.

Move will move the object or group around the area – it will not allow objects to occupy the same space so other objects that are in the way of the movement have to be steered around.

Whilst in Edit Object mode, the currently selected object can be changed by pointing the arrow on the new object (in the view) and clicking – easy isn't it?

Bring the cube down to the floor and then re-size it so that it ends up near position 4000,30,6250 and close to size 700,400,450. This task will be easier if you set the size first.

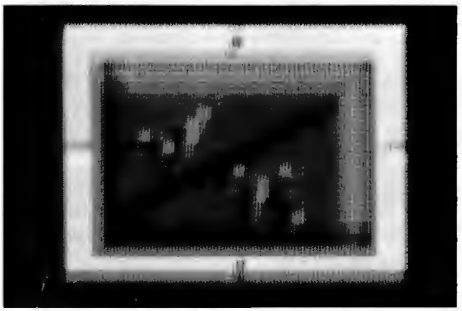


Figure 3.4

Now let's tidy the whole thing up. Return to the main control panel by clicking on OKAY, then click on the fourth icon button (as shown in fig. 3.4) and select the cube. This icon allows us to fine tune the object's **attributes.** These are its characteristics and abilities and we will be looking at them in depth in the next section.

For now, we are interested in the Position and Size variables. Click on the number and a cursor (flashing black square) appears at the start of it. Now type in the number that you wish to set this variable to and press enter. By working down the list you can position the cube at exactly 4000,30,6250 and size it to exactly 700,400,450. Do this and then click on OK to remove the menu and return to the main screen.

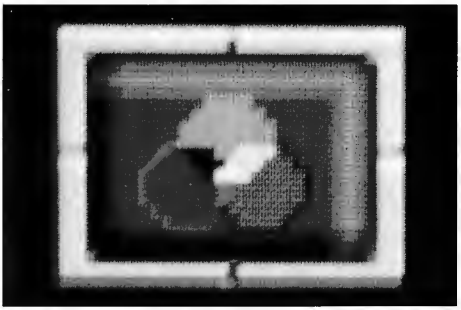


Figure 3.5

Now we'll colour the cube. Click on Icon number five, it's three coloured circles as shown in fig. 3.5. Select the cube and then click on OK.

Again the bottom of the screen changes to give us a new control panel. This is the **colouring** section of the editor. You are presented with a palette of 256 small coloured blocks, a window on the left divided into six coloured blocks, a couple of buttons, and another window on the right with a single coloured block in it and a number underneath.

As with the object editor you can select a different object by clicking on it in the view. The background can also be selected this time and it's colours changed.

Basically when we are using the colour editor we pick up a colour with the left mouse button and paint with the right. Colours can be picked up from objects in view, or from any of the boxes on the palette, or in the grid on the left. The window on the right shows the current colour and colour number – the paint currently on the brush.

The first colour (000) in the palette has a special use. It is invisible paint – it tells the computer not to bother drawing this face of the object.

The window on the left shows the current colours of the sides of the selected object. You can change the colour of the object by painting these squares, and these squares will change when you paint the object. For the background, the top colour is the sky and the bottom one the ground. Try painting the cube so that the faces are coloured (left to right, top to bottom, in the left window) numbers 14, 14, 15, 15, 0, 0. The bottom face is in contact with the floor so there's no need to see it, and we're going to put a roof on the top so there's no need to draw that either.

Now it's time for some practice at what we've learnt. We're going to build a house. The roof will be a pyramid, the door will be a rectangle, and the windows will be a hexagon,

two pentagons, a triangle, and four more rectangles. Create the objects in this order and then stick them on the house and colour them. The positions, sizes, and colours I used are as follows:

Object	Position	Size	Colours
03	3975,430,6220	750,275,510	13,13,12,12,15,0
04	4440,30,6250	165,305,0	0,10
05	4530,240,6700	90,105,0	0,4
06	4700,220,6560	0,90,90	0,4
07	4700,290,6455	0,90,90	0,4
08	4000,330,6280	0,75,210	0,8
09	4060,200,6700	190,180,0	0,8
10	4260,200,6700	100,175,0	0,8
11	4095,185,6250	200,150,0	0,8
12	4000,195,6290	0,130,195	0,8

To close the top of the pyramid, the corner points must be manipulated. This is done in the **point** section of the object editor. The point will be surrounded by a flashing wire-frame cube and can be changed by clicking on "next" and moved with the arrowed buttons as with all other functions on the editor.

Once the house is finished, colour the sky black, the ground deep green, and the floor brown.

So far in this section of the tutorial we have been creating and manipulating *Objects*. Once you have read, tried, and understood this section you will be ready to create and use some groups.

Select the Create Object Icon Button (first one along – picture of a cube on it) and then click on the first item in the bottom row of potential objects. This will create a "group". A **group** has no physical form, it is a "folder" in which we can stick all the objects to duplicate and manipulate them. We will move and duplicate our house so rather than do this to each object individually, we will put them all in a group and save time.

The group is created when you click on its icon (the stack of cubes in the Create Object Menu). If you now select delete object (second button – cube with a cross through it) and have a look at the object list (the other way to look at the object list is to

click on the gloved hand icon, the eighth one along) you will see that the group is on this list. Now we must add all these objects into the group. We do this by editing the attributes of the group. Click on the fourth icon. It's a tick and a cross. We'll be examining this button in depth next section. On the object list select the group – this is very important so don't mess up here.

When you click on OK a new panel appears. This is the Edit Group panel. It gives the name of the group, its number, its position and size, two buttons (remove and add) and an empty list window.

To add an object to the group click on **add**. Do this now. You are presented with the object list. Select the cube that forms the body of the house. Click on OK. The cube is added to the group. Add all the objects in the object list to the group *except the floor*. The list window in the edit group panel fills up. Once you have them all in the group, click on OK to get back to the control panel.

Our house can now be moved around. Click on the sixth icon along (the box with the red extension). Select the group from the object list. You can now move and turn the group. Position the house where you want it and then let's make another.

Return to the control panel. Click on the seventh button along. It's the **duplicate object** icon, and, as you might expect, it duplicates objects, and, more usefully, groups.

Select the group from the object list. You are asked to select a duplicate position. You can enter the exact co-ordinates you would like, or you can simply place it on any available side of the existing group. Select right (by clicking on it) and then click on OK.

"Voila!", as the French rarely say, another house appears to the right of the original one (if things were this easy we could solve the homeless problem overnight, but sadly things get more complex in the real world, sigh, never mind).

On the object list you will notice that both the group and the objects have been duplicated. Play around with what we have learned this section of the tutorial and when you have objects and groups down pat, give yourself a cup of coffee and meet me at the start of section 4.

4: Scock, you've...GOT...to get us out of here. Or: Beam me up Spotty. (Surely there's some mistake here. Ed.)

Contrary to all previous knowledge, I can exclusively reveal at this point that teleportation is here, now. It exists, we can use it, and it is found in the simulated worlds generated by 3D Construction Kit 2.0.

Unfortunately, we do not yet possess the knowledge to get a real human being into the world created by this program, but with just a few more years research, and some powerful hallucinogens, a solution could soon be at hand.

Turn on the machine and load up 3D Construction Kit 2.0 so that you are presented with the familiar (and, by now, somewhat predictable) opening horizon in area one. Create a cube, and lower it so that it rests on the floor, enlarge it to about 250 by 400 by 250. Walk into it. Now rub your bruised nose and pick yourself up from the floor. Normally, the program will not allow us to walk through objects (this can be changed – more later).

Any object, however, can be a **Teleporter**, and this (and other object attributes) is the subject of this section.

Teleporters transport the player to **Entrances**. Any area can have one or more entrances, and they are created in the Area menu. Let's do this now. Move a long way back from the cube and view it from an angle so that you are looking at a corner, try Location (1967,310,2398) Rotation (0,45,0). Now click on Area and click on Create Entrance. Right, you've done it. Your current location is defined as entrance one. Let's check this works now by moving away from this location. Go right a bit and then follow Area – Edit Entrance. This box allows us to look at the entrances created in this area. The top line gives the number and name of the entrance. The next two rows give the position and rotation of the entrance (this will be the same as your position when the entrance was created. All of these fields can be edited in the usual way.

The next row has three buttons. **View** will show you the view from the entrance, **Goto** will move your current position to that of the entrance, and **Set Current** will set the entrance location as your current position (rather like re-creating it). The **Prev**(ious) and **Next** buttons are dimmed because we have only created one entrance, but if we had more than one they would allow us to move between them. Any changes that we make will not be permanent until we press **Store**. Clicking on **Okay** will return us to the main editor.

Now that we've got an entrance, lets teleport to it.

There are two ways of making an object into a teleporter. We're going to look at both of them in this section but we'll do the easy one first. Click on Icon Four (the tick and the cross) and select the cube. The object attributes appear. We've already looked at this screen when we were setting the precise location of our objects but now we're going to look at the set of ten buttons on the right of the screen.

These are the objects main **attributes** that tell 3D Construction Kit 2.0 how an object is going to behave. Objects can do lots of different things, they can be visible or invisible, they can be drawn in different ways, they can teleport and sense, they can even move around. The more things an object can do, the more memory it takes to store it. The last few lines of the text window on the control panel detail memory and you can see it go down as you work!! These attributes that we will be setting here tell the computer that the object is able to do these things. It does not make them do it, or give details of what will happen, it simply makes them able to start moving, or sense things, or whatever.

Let's look down these buttons:

INV – This makes the object invisible. When an object is invisible the player can walk through it and it will not appear in the view window. The object is not active, (ie it can not still sense and teleport).

DES – This makes the object destroyed. This is the same as invisible but the object is now neither visible nor functional. Both of the these attributes can be set during play (for example, you can tell an object that if it is shot then it is destroyed and upon shooting by the player it will change to destroyed.... you can also make invisible objects visible, thus causing them to appear in the view window).

WIR – Tells the computer to simply draw a wire frame of the object, and not to colour the faces in. This speeds things up and can be used to dramatic effect. The player cannot move through a wire frame object, even though it may appear to be full of space.

TAN – Controls an object's tangibility. When this icon is switched on the player can walk through an object although it will appear to be no different from any other (great for secret doors and hidden locations).

MOV – Enables an object to move. Any attempt to animate an object whose MOV flag has not been set will only end in tears and irritation. Remember, this does not tell an object how to move, or when to move, it simply tells the computer that it can move.

SNS – Makes an object a sensor. The sensor button at the bottom of the screen becomes usable. More on sensors in the next section.

CLR – Sets an object as colourable. This means that it's colour is changeable during play. If this is not set, then the object will not be allowed to change colour.

ALW – When in normal play, the game will not execute **object conditions** (special instructions relating specifically to a given object) unless that object is collided with, shot, or activated. When the ALWays attribute is set then these conditions will always be checked (great for objects set to self destruct after a certain amount of time, or always moving).

LOC – Locks the object off from being animated.

TRN – Makes the object a transporter (at last!).

Click on TRN. It toggles on, and the **Transporter button** at the bottom of the screen becomes solid. Click on this now. We are in area one, and the entrance we have created is entrance one. Type these numbers in at the relevant point then click on okay to get back to the attribute screen, and okay again to get back to the control panel. Now walk into the cube. As you impact with it you are teleported to the entrance we created earlier. Now that's flashy.

Teleporters are not simply useful for travelling around the area however, as you have deduced, they can be used to go from area to area. To have a go at this we must create a new area. *Follow Area – Create Area.* A new area is created and we are dropped in it. Let's change the colours in this area. The Area Colours that is. Click on Area again and then select **Area Colours**. The default colours are shown from the palette (16 colours on an Amiga, many hundreds on a PC). Select a colour by clicking on it. The colour bars move and can be moved to change the colour. Press **copy** and then select a new colour: the currently selected colour is copied (wow!). Press **reset** and the colours are set back to their states before you started messing around. Select two colours a little way apart. Click on the first. Click on **spread**. When you click on the second, the colours between them will be set as stages between the first and the second (this is a lot clearer when you do it – so do it). **Cancel** returns you to the control panel and resets before leaving. **Okay** sets these colours and returns to the control panel. Try messing about with these and then return to the control panel and lets get on with things.

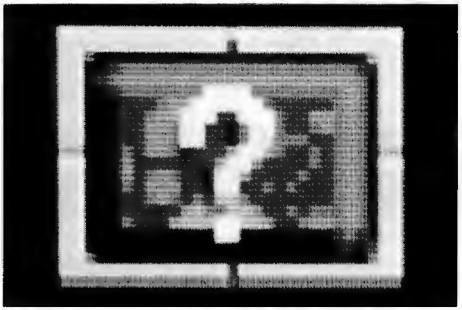


Figure 4.1

Create a pyramid, enlarge it to about 250 by 450 by 250, and bring it down to ground level. Set an entrance on this area at location (5832, 280,2830) rotation (0,320,0). Now we want to make the pyramid into a teleporter. We will do this by editing the **object conditions**. Click on the third icon on the control panel pictured in fig. 4.1 – it's the one with the large question mark on it. Select the pyramid. A very ominous large grey screen appears. There is a cursor in the top left hand corner of the main page – this is the page where we program each object, tell it what to do. Objects understand only one language, and that language is called **F.C.L. (Freescape Command Language)**. The system which the computer uses to draw all of the views is Freescape, the areas are known as Freescape areas, the objects as Freescape objects, and so on until we run out of examples or lifespan.

Fortunately FCL is a very logical language to use, and it is very easy to learn. It would be foolish to explain and demonstrate every FCL command in all of its permutations in this tutorial, but we will use most of them, and by the time you have finished you will have a clear idea of the logic of FCL and will be able to extract maximum understanding from the FCL Reference Guide, which is later in this book.

We want our teleporter to teleport us to area one when we collide with it so this is what we must tell the object.

Type the following into the machine:

if collided? then goto (1,1).

Believe it or not (I said FCL was easy didn't I?) this means "If this object is collided with then go to entrance one in area one." Got it – if not then check out an english language difficulty. The numbers are put in brackets, separated by commas, and known as parameters. Parameters can be optional. For example, if the command had read:

if collided? then goto (1)

then the program would take us to entrance one in the *current area.*

Once you have typed this in click on OK. The program will **compile** the command (check you have typed it in correctly and tag it on to the object). If there are any errors you will be told about them now.

Naturally there are no errors in our lines of **code** (flashy jargon term for the words we type in – well worth learning for the truly "in the know") so we are returned to the control panel – now we can test it out: walk into the pyramid.

We are briskly beamed across the freescape cosmos to entrance one at area one. Now, by editing the commands to the cube in either way, set the cube to zap us back to entrance one in area two. The way to do it is printed below (for the truly slow learner):

Follow Conditions – Object Conditions > – *Edit* and enter the code:

if collided? then goto (1,2)

and click on OK.

By pressing forwards we can now walk endlessly from world to world being teleported more times than Captain Jerk. Right, that's the end of lesson 4 : you can take a teabreak/cigarette break/sleep break now and then meet me on the next page for section 5.

5: Holy Exploding Cubes Batman!!

By now, you will have noticed that when you have the arrow on the view window and you click the left buttons four flashing lines appear from the corners and converge at the arrow. To the more violent among us this will be immediately recognisable as shooting. New Age pacifists can take this to be sending out rays of love to pacify and re-mould the negativity of the location, but the program calls it shooting, so it's probably best to use that.

There are three things that the player can to to an object (stop smirking – that isn't one of them). The player can **shoot** it, the player can **activate** it, or the player can **collide** with it. To test for each of these we use the FCL command "If" in conjunction with the words "shot?", "collided?" and "activated?" to test for shooting, collision, and activation respectively.

At this point you begin to realise that FCL is not a very complex language, in fact, it's basically English (or American, depending on whether you say "sidewalk" or "pavement"), with a lot of the words missing (especially the little ones that are no good in charades because everyone gets them too quickly).

Get 3D Construction Kit 2.0 running and create five rectangles in area one. Leave them at their original size and arrange them so they form a cube on the ground (the bottom face is the one you should leave out. Colour them if you want to... a black background looks great for this one.

Now let's start programming the objects to move. First we have to get into the attributes – icon four – and set all the rectangles to movable by clicking on MOV in the attributes columns.

Next we have to decide when we want them to move (when activated, when collided with, or when shot). We'll make our one move when shot. The effect we are trying to get is that of the box opening up by exploding from within so we must tell the computer to move all the sides of the cube in different directions. The instructions that tell the computer how each item is going to move is called an **animator** and we will need five of them (one for each face of the "cube"). There are quicker and easier ways to do this task than the one we are going to use. We could for example make all of the objects into a group and deal with that group, but we will take the simplest route, as this is a 3D Construction Kit 2.0 tutorial, not a book on advanced FCL technique. Also, going the long way affords us an insight into what is happening and why.

Create five animators. *Follow Conditions – Animators > – Create* to create each one. Now follow *Conditions – Animators > – Edit* to begin editing the conditions (a brief

reminder – conditions are the groups of instructions that tell the computer what to do... this is all explained in the reference manual later on anyway... if you want to get in that deep, why don't you read that instead... of course, it won't have this tome's witty, self-mocking style, but that could just as easily be seen as an advantage. Anyway, back to reality:) the screen asks you to select a condition – select number one.

An animator comes in two parts, the bit that tells the program what objects to move, and the bit that actually moves them. Type in the following code:

```
Include (3)
Start
Loop (30)
Move (0,0,10)
Again
End
```

The first line tells the computer to include object number three in this animator. There could easily be a list of objects here if this animator is to be used to move more than one thing, they would be included as separate includes... don't worry, I'll clarify that... a typical start might be:

```
Include (10)
Include (4)
Include (2)
Include (26)
etc...
```

You cannot express this as "Include (10,4,2,26)". I know it's a shame, but those are the breaks folks.

The second line of our code is "Start". This tells the machine that the list of objects is over, and the movement commands are about to begin.

The next bit, "Loop (30)" means "we're going to do this next bit 30 times". The computer will come back to the command "Loop (30)" when it is told to start again and will continue to do so until it has done it 30 times. The number in the bracket can be changed.

Move (0,0,10) tells the computer to perform a relative move in each of the axes of the amount specified by the three numbers – so this line moves the included object (number three) ten units in the positive on the Z-axis. It adds ten units to the object's Z co-ordinate.

Why the Z co-ordinate? On my cube object three is the face at the back of the cube – to move it away from the cube I have to move it on the Z axis. On your cube, the faces may all be in different positions. Now the fun starts – You have five animators and five objects. See if you can write the code for all the animators. Each animator will be essentially the same. All you will need to change is the movement co-ordinates (minus numbers, such as -10, will move the object towards the zero point), and the number in the Include brackets.

This type of movement is known as **relative movement**. The object is moved an amount relative to its current position. The other form of movement available through FCL is absolute movement. In **absolute movement** the numbers in brackets give the co-ordinates to which the object will be moved. The commands for making absolute moves are detailed in the reference manual, which, as you may be realising, will have to be digested in full with gravy and two vegetables before you get the maximum out of this rather horrifyingly complex program that you thought "might be a bit of fun". Still, you've got this far, and that's a very good sign that you'll go the distance. It all gets flashy from this point onwards.

Right, while I've been talking rubbish, you should have got those animators written. If not, then put this down and give it a try. There are very few mistakes to be made, so just get on with the working out and typing in (you can find out which object is which by reading the text window and clicking on them as you move around the box).

Have you done it? Really?? You're not just reading on because you think you're smart so you know what's going on?? You're not on a train are you?? Reading this in a shop?? You must have been here hours!!

Okay, I'll take your word for it that the jobs been done so we'll test them out. We need to write object conditions now so click on the question mark icon and select whichever object is Included in animator one. You must have written it down somewhere by now.

Here's the code. I'll explain it after you've read it, but see if you can work it out for yourself:

```
If shot? then
Startanim (1)
Startanim (2)
Startanim (3)
Startanim (4)
Startanim (5)
Endif
```

This piece of code asks the computer if the object has been shot, and if it has then all the code up to the "Endif" command is executed. We start all of the animators in this piece of code. If we didn't only the side of the box that we had shot would move. This code is exactly the same for each of the five sides of the box, so type it in to the object conditions of each of them and then meet me back at the control panel.

Get a nice corner view of the box and shoot it. The box should "explode" open and then stop with the top hanging in the air. We could make the top land by lengthening its animator, or we could simply make the objects "destroyed" (using the command Destroy – no great shocks here either) so that they vanish. Play around with the instructions to your heart's content.

You will notice that once the cube has been shot open it doesn't do much. To put it back to its initial state click on the reset (exclamation mark in a red triangle) icon. Try setting the cube to open on activation instead of being shot (you activate an object with the right mouse button). Try and bring the top face of the cube back down again (this is not as easy as you might think and is a damn good FCL programming exercise).

For the moment have a bit of fun with FCL, and then move on to the next section of the tutorial. We're two thirds of the way there now and things get more and more visual from here on in. FCL is covered in more detail a little later in the manual.

6: Faders and Sensors and Making your own Videos

There's a bit of the object attribute screen we haven't looked at yet. It's the button next to the word **Fade**. Fade can be set in four different ways and pressing the button moves us through the list of fade effects. **Off** means the object does not fade. **In** means the object fades in when it is first made visible. **Out** means the object fades out and becomes invisible. **Bounce** means the object fades in and out constantly.

The number next to the word "Fadeval" tells the computer how **transparent** an object is. If an object is fully transparent (Fadeval = 9) then it will not be drawn – but you will still be able to collide with it. Fadeval is useful for windows, screens, etc., and the **mask** (which bits are drawn and which bits are not) is definable. Fadevals number 0 to 9 are the stages through which an object is faded in or out and are fixed in to the system. Fadevals 10-15, are user definable. To understand the way Fadeval works set Fade to OFF and try setting Fadeval to different numbers between 0 and 9 and observing the results in the view window. You can define your own masks (the system calls them **user fades**) by following the path General – Edit User Fades which allows you access to the fade editor. Select the mask you want to edit by clicking on its number and then click on any of the blocks in the eight rows and columns to toggle them on or off. You can use your Fadeval in an object, by setting the object's Fadeval to the number of the user fade that you have defined. Any black spots on your mask will be invisible points on your object, any grey spots will be drawn.

Once you've played around with fades and the fade editor you will notice the way that when an object fades out it sets itself invisible – this is especially useful when the fade is called by the FCL command "Fadeout" which might be used in the following way:

Edit the cube's object conditions (the question mark icon). Enter the following code:

If activated? then fadeout (2,1)

The (2,1) means object two in area one. Leaving out the second number means in the current area.

Try activating the object and it will fade out beautifully (or badly, depending on which fade you use!).

Right, that's fading – now lets see what other objects can do to you – we need **sensors** in our land, and this is how we do it...

Any object can be a sensor, or you can create one by clicking on the little radar dish icon in the "create objects" menu.

Let's put a sensor next to the cube, using a pyramid. Create a pyramid and drop it to the ground next to the cube. If the cube has faded out bring it back with the reset button (bottom right, and this is the last time I'm telling you).

Now edit the attributes of the cube (icon four, the tick and the cross). Click on SNS. This makes the object a sensor. The sensor button at the bottom becomes usable, so let's get going....

On clicking this button we are presented with a screen which shows us in which directions the sensor is active (these can be toggled on and off, they all start on), the offset from the centre of the object where the sensor is placed that marks the centre of its sphere of sense (and if you understand that you're a better person than I am), the range of the sensor (in units), it's speed (how many 50th's of a second it waits before reacting), the sound that is to be played when the sensor is triggered, the procedure to be executed (if any), and the type of sensor that this is.

There are two types of sensor, a detector, and a shooter. We'll pick a shooter because it gives a visual indication when it is triggered (detectors are used when you don't want the player to know they've been sensed). Make sure this button is set to Shoot.

Back in the paragraph before last I managed to sneak in the word **procedure** without anyone knowing, but now that I have admitted my cunning plan I'd better explain what a procedure is.

A procedure is a section of FCL code, like an animator, or a condition, that can be called by any other part of the code. In the same way that we created the animators but they were not run until the machine reached a "StartAnim" command, procedures can be complex series of instructions that are called from anywhere in the program by the command **proc** which is always followed by the procedure number in brackets. Thus, a sensor being triggered can call a procedure (if the number is set to 0 no procedure is called) which closes doors, or, in this case, fades objects out. We will set our sensor to fade out the cube when it is triggered.

Set PROC to 1, and set the sensor to be of type "detect" and range 500 and then follow *Conditions – Procedures – Create*, and then *Conditions – Procedures – Edit* and select procedure number one.

This procedure will be one line long – it reads: Fadeout (2,1)

Type it in, then get back to the control panel and trigger the sensor. The sensor detects, the cube fades – poetry in motion – this is the stuff that is made of.

Now then, there may be points in your game when you want to move the player around the area (when the player is standing next to something dangerous or when the player arrives in a new area for the first time, for example). When you want the land to move without the player controlling the movement, you simply stop the game and show the player a video. The best thing is, you don't need a video recorder... 3D Construction Kit 2.0 comes with one built in... here's how you use it...

Click on General on the menu bar and then click on **video**. The menu that unfolds shows us the four options available in the video section. Before we can record a sequence of video, we need to create it. Do this by clicking on **Create Sequence**. Now do the same thing again but select **Record Sequence**. This is where we do the majority of the work. Select Viewpath number one (it's the only one on the list, the one you created) and the control panel changes to reveal a video recording panel.

The numbers on the left give us the information we need. The first number is the number of moves made by you during the video. The second number is the length of the video in minutes, seconds, and hundreths of a second. Below this are the name and number of the current video.

Okay, before we go any further, it's time to come clean. Although we call what you are about to record a video, it isn't in the strictest sense. The computer is about to record your actions – not the scenery that passes before your eyes. In other words, if you want the video to look the same every time, you will have to be sure that the video starts playing from the same player position and rotation every time (fortunately there have been controls provided for just such an eventuality).

The numbers on the right hand side of the vertical blue line are the current timer position of the recording (time elapsed since recording/playback began), and below this the current move being performed (its number in the sequence of events).

When you first arrive at this control panel all of the numbers will be set to zero with the exception of the video number which will be set to 001. To the right of all the information are the control buttons. The first button (blue square) is the **stop** button, next to it is the **record** button – this is one of those neat videos with single touch record facilities – and third along is the **return to start** button. The fourth button is the **play** button, and the fifth button erases everything *from the current play position onwards.*

Click on record and then when the tape is rolling (numbers changing in top right number display) move around the area a bit, turn round a few times, and then click on

stop. Click on rewind, then press play. Your last few moves will be repeated: each move will take as long as it took you to enter it initially – the sequence of events will take precisely the same time as when you first performed it. This type of playback is called **realtime** playback. The other type of playback supported by 3D Construction Kit 2.0 is **normal** playback. When in normal playback, the video still executes every move you made, in the right order, but it does not wait at the end of any move. Using normal playback you can create some truly spectacular sensations, for example, it could be used to knock the player around when they are shot – a simple wiggle around which returns to the same position as it starts would achieve this.

To choose playback mode when you have finished recording, click on Exit and then follow *General – Video > – Edit Sequence* and select the current viewpath.

In this control panel you can set the name of the video sequence (or **vlewpath**), its playback mode (toggles between realtime and normal) and also set it to repeat or play once (I assume that this needs no explanation – if you feel I am wrong please feel free to call the Government Department for the Profoundly Stupid. If you can find the 'phone, that is).

The repeat count tells the program how many times to repeat the sequence. Zero causes it to repeat infinitely. It is in this panel that you can also set the start position for the video. This position can be specified in terms of an entrance, or in the specific X,Y and Z co-ordinates and rotations. To tell the program which one of these start positions to use you click on the bottom button (start mode). This will toggle you between **current pos** (performs the moves in the video from whatever position the player is in when it is started – useful for effects of the type described above), **entrance** (as shown on screen), and **position** (as defined on the panel).

To check that you are happy with your creation, follow *General – Video > – Record Sequence* and you can view the video by clicking on play (the triangle). These videos can be used as part of conditions by using FCL. For further details of the FCL commands consult the reference guide (which does, for those who were worried, contain full details of FCL, its form, and its uses).

Well, only one more section to go and you'll be using 3D Construction Kit 2.0 like a pro. You've already been shown everything you need to create a world – in the next few pages I'll take a walk with you through the General menu, and we'll learn how to turn the world into a game.

7 : A Player of Games, and a few loose ends

For those who are, perhaps, slower on the uptake than most, it needs to be pointed out that games created with 3D Construction Kit 2.0 are going to be of the "moving around inside a three dimensional space" type. Although Space Invaders (gosh, it's back again... how did it ever get to be such a yardstick?) would be most interesting in 3D it would certainly not have been such an immediate hit if it had been written this way.

Players of games written with 3D Construction Kit 2.0 will need access to certain things: controls, the view, instruments which relay information (like a clock, or an altitude meter), and snippets of information in text form (like "You have crashed, idiot." or "Another try wimp?" – you know the sort of macho posturing stuff I mean).

3D Construction Kit 2.0 gives you access to all of these things both through FCL and through a set of commands in the editor which will be the subject of this section.

3D Construction Kit 2.0 is not an art package. To design your playing screens and title screens you will need to use an art package. Fortunately 3D Construction Kit 2.0 supports nearly every popular art package and will load screens of different types created on different packages. The playing screen is the border around the view window on which we place the control buttons and the instruments. The program knows these playing screens as borders and we have included several demo borders with the program. The one we are going to use is called "Driller.IFF". Follow *File – Borders – Locate* and then tell the machine the name of the disk folder in which the borders for the game are to be saved (the folder we are using for this tutorial is on the second disk and is called "BORDERS" (not an altogether original title but a functional one nonetheless). Then use *File – Borders – Add* to add the "Driller.IFF" demo border to the program. Make sure that you set the Freescape Window button to "Off" if it is not already set that way otherwise the program will not load any border art that covers the screen area where the view window is currently set. Set the colours to come from the border (this means the freescape view will be coloured with the colours in the border. Obviously, when you put a game together yourself you will bear this in mind when designing the freescape world, but this is just a tutorial dammit! What do you want? Blood? Red blood??) The border will be loaded into the borders list as Border 001. Let's have a look at it. Follow *File – Borders – View* (selecting Border 001). The disk drive whirrs (or purrs if you are rich enough to own a hard drive) and the screen appears. Although I don't want you to do this yet, you can click either of the mouse buttons to get back to the control panel.

Let's take a look at this border. Firstly we have to remember that this is purely a piece of artwork; none of the buttons work, none of the dials work, they are all purely graphical representations for the player. If we want instruments and buttons that work, or even if we want to put the 3D Construction Kit 2.0 view into the window drawn on

this page (centre top) we need to do that ourselves – That's what I'm about to show you... so let's get started.

Return to the control panel (have you noticed how many times the words "Control Panel" crop up in this tutorial?). Let's set the view window.

Follow *General – Set View Window.* You are presented with a panel that tells you the position (X and Y co-ordinates *on the screen*) of the bottom left hand corner of the view window (zero point is bottom left) and the size of the window. You can change the size of the window by entering in numbers and then clicking on view to see the window. However, we want to set it to the border we have loaded, so click on set to set the window in a more visual way.

You will see a large box (the view window) with a small box in it's bottom right hand corner. You will also see a panel that details the position and size of the view window. Move the box by holding the left mouse button down whilst inside the box and moving the mouse. Change the size of the box by holding the left button down and moving the mouse whilst in the tiny box (this moves the corner of the window). Try this and once you've got the hang of it meet me at the top of the next paragraph.

We need to set our window to fit our border, and this would be easier if we could see the border. Click on the small panel. The borders menu appears and you can select our border. Once you've selected it click on OK. Set the view window within the large front screen on the border then click on the right button to return to the view window panel and on OK to return to the control panel.

Now, finally, after much tension and rumour, the meaning of the last of the icon buttons. A simple process of elimination will tell you that the icon button in question is the one with the eye on it. This button puts the whole system into test mode (and, in case you've been panicking, pressing the F1 button will return you to the editor). When you click on the eye button the screen will clear and the view window you have just set will appear. Do this now.

You are now in at the playing end of the system. The left button fires (just like normal) and the right button activates. It's not really very exciting is it – in fact it looks like the border has been forgotten, and there seems to be no way to move around the world. Let's deal with these problems one at a time.

Follow *General – Defaults* and you will be presented with a new panel. This panel is fully explained in the Menus section of the reference guide but we must learn now what a default is and how they are used in 3D Construction Kit 2.0.

Defaults are the way that (1): the game will start, and (2): the game will *reset* itself. To perform a reset whilst in playing mode simply press escape. If a border is defined on this panel it will be loaded. When you first go into test mode the game does not reset itself – you are put into the world at your current position in the editor, and no borders are loaded (there is a good reason for this – borders take time to load, especially if you're too poor for a hard drive, and when testing a simple part of the game we do not want to have to wait around).

We want our game to start with the border we have loaded (number one on the borders list), so enter this at the bottom of the panel and then click on OK.

Now go into test mode and reset the game (by pressing escape, the "esc" button on the top left of the keyboard). The view window can now be seen in the border, but we still have no controls or instruments working. We'll do a couple of each so that you get the idea and then, for your final piece of homework (ugh! bad word!!) try and get the control panel fully functional.

We'll set some controls first. Press F1 to get back to the control panel and then follow *General – Controls*. Yet another dazzling panel appears before our somewhat tired eyes (very tired if you've done this whole tutorial in one long masochistic session). Many of the controls in the game are already defined. Each **control** has a number, a position for an on screen button, a definition of which mouse button applies on this screen button, a keyboard control for the command, and the function of the control. Move through the controls with next and when you've had a look at what is already available you'll see that the programmers have already included nearly everything the average game could want. However, if your game contains a special new command to make the toast, you would simply add a new control, and tell the editor which procedure (a short set of commands – maybe to run an animator, or blow up a cube, or anything really) to run when this control is activated.

Setting a button or two will make things clearer. We'll set the rotate left and right buttons, and the move forwards.

Select control number one, "Move Forwards". Click on set. The tiny box in the top left hand corner is the control. Click on the co-ordinate panel and load up the border, then set the button over the forwards arrow on the icon panel at the bottom of the border. This done, click on the right mouse button. Find controls number seven and eight and set them on the correct sides of control one. This done, click on OK and then kick the program into test mode (the eye icon – last reminder). Do a reset ("esc" key) and try the whole thing out. Not very exciting without things to move around, so load up object number 0001 from the clip art folder supplied with 3D Construction Kit 2.0. Set it to load in the position as saved, and then pop into test mode and move around it. Pretty cool eh? Well, maybe not, but it's certainly pretty locomotive, using the word as a verb of course.

Now we'll set up a simple instrument, it will be a clock of sorts that tells us how many minutes we have been playing. We'll use a **numerical** instrument (one that displays numbers) for the minutes, and we'll use a dial for the seconds. Go back to the main control panel and follow *General – Instruments – Create.* Do this twice as we want two controls (of course you knew that didn't you!), make the first one a numerical instrument, and the second one a dial. Now follow *General – Instruments – Edit.*

Don't panic! Click on set and position the numerical instrument in the black box in the bottom left of the border. Return to the panel by clicking on the right mouse button when the instrument is set. Click on **store** *before you move on to set the dial* otherwise all the changes you have made to this instrument will be lost. This dial will count minutes and will take its value from one of the variables available to the user in procedures. We'll use variable 53 (because I like that number) for the numerical counter and variable 54 for the dial which will count the seconds. On the panel set the variable to 53. The two values underneath define the range of the instrument, which in this case will be between zero and 59 (because, for various reasons way too irrelevant to deal with here, there are 60 minutes in an hour). This will also be the range for the dial. Set Value1 to zero (it should already be zero if you haven't been playing around!) and set Value2 to 59. *Store* your work and then set the dial up in the left side panel of the border (next to the view window – you will have to make it a long thin dial). Set all the variables on this instrument (fifty four, zero, and fifty nine top to bottom) and, after you've stored the result, return to the control panel.

To write the code to move the dials we need to create a **global condition**, a condition that will always execute no matter where in the world we are (such as time running forwards). Other types of conditions are detailed elsewhere, so for now follow *Conditions – General Conditions – Create*, and then follow *Conditions – General Conditions – Edit*, and select the only general condition available.

A familiar window appears. Let's write some code.

There is a command in FCL designed specifically to handle time controls. Remember, we do not alter the instruments in the code, we simply change the variable where their information is stored – 3D Construction Kit 2.0 does the rest. The command we are looking for is called **TIME**. Yet again we are disappointed at the surprise value of the command word, but can reflect upon its aptness for many days.

The code you need to type in looks like this:

```
Time (v52,v53,v54)
```

This tells the computer to look at its internal clock and store the hours in variable 52, the minutes in variable 53 (which feeds our numerical instrument) and the seconds in

variable 54 (which sets the dial). If we wanted to, we could now go on to define instrument three as an hour counter or dial and set it to variable. The command "Time" needs three variable numbers in brackets after the word, but, as we have demonstrated, a spare variable can be used.

When constructing a large program it is advisable to have a pen and paper constantly ready to note down variable numbers and the like, as this can save much tearing of hair and gnashing of teeth at a later stage.

Once you've got the code in, click on okay and then enter test mode and perform a reset. If the dials don't work then go back and check your code and whether you remembered to store the values you entered before clicking on OK in edit instrument. The other values on this panel (**col1**, **col2** and **font**) refer to the foreground colour (text or hands on a dial, or colour of growing bar on a horizontal or vertical dial), the background colour, and the type of text (-2 is larger than -1, other fonts are to be released at a later date) respectively.

That's just about everything you're going to need to create a seriously complicated game, but for those who like everything animated and moving, we're going to look at one last feature of 3D Construction Kit 2.0 that enables you to give your games the professional edge. I'm referring to brushes, and brush animations.

A **brush** is a little piece of art, cut from a border, or any screen generated in an art package. Put several of these little pictures one after another and cycle through the series and the whole bundle is known as a **brush animation**. There's an excellent example on the disks supplied with this program. It's called "World" and it's in a folder called "Brush". Prepare to complete your formal education by following *General – Brushes > – Cut Brush*. Find the brush and click on OK. A picture loads – it is a picture of many worlds stacked next to each other with a few large Frame Notes. Up at the top left hand corner of the screen is a drag box – the same as we used to set the view window, controls, etc. The position and size of the box are shown on the small panel. Position the box over the first globe (just testing – you can see it's already in the right place). The box, by an uncanny coincidence needs to be set to size 32 X 32 for this brush (remember, a brush is a small picture that can be any size).

Cut the first brush by going to co-ordinates 000,000 and clicking the right button. Now repeat the procedure and, for time's sake, cut out every second frame working left to right, top to bottom. Your X co-ordinates will be 000, 066, 132 etc and your Y co-ordinates will be 000, 032, 064, etc. When you have cut 11 brushes and returned to the control panel we will continue.

You can check each brush is cut correctly by following *General – Brushes > – Edit Brush*. This panel will also give details of the brush size and allow you to rename it.

Cycle through the brushes using **pre**vious and **next** and if a brush has been cut wrongly delete it (following *General – Brushes > – Delete Brush*) and re-cut it.

Now we are going to animate these brushes by cycling them so that the world appears to turn; hopefully, in the process, we can put this flat world theory to rest once and for all and move on to some more constructive human endeavours. Click on General, then on **Brush Animations**. Create an animation (no prizes for guessing that this is done by clicking on **Create**) and then follow *General – Brush Animations > – Edit* . There will only be one entry on the list: select it.

The panel we are presented with tells us the name and number of the brush animation, its status (on or off, but in more high-tech words), it's on screen position (it will be laid on top of the border, remember), it's speed and length (currently zero because we haven't put any brushes in it), and its mode. There are two buttons next to the title **Initial Mode**. The first button has six states. They are as follows: Single will play the sequence of brushes once and then stop; repeat will cause the sequence to play again and again; bounce will cause the sequence to play start to finish and then finish to start, repeating infinitely; random means that brushes from the sequence will be played randomly one after the other; inactive means that the brush is not drawn; stopped means the animation does not change but the current frame remains on screen. The other button tells 3D Construction Kit 2.0 to play the sequence either forwards or backwards. Set these buttons to repeat forwards and then click on edit.

This new panel is very similar to the edit group attributes panel. Add the brushes to the animation list (in the same way as you add objects to a group) and put them in numerical order. Put all eleven in. When you've done this click on OK.

Now click on **preview**. The new panel shows your brush animation playing. This is where you test the different speeds and modes. Play around with the edit panel and check the results using preview. This way you will gain a thorough understanding of brushes and brush animations.

Once you have set your brush animation running at a speed you like set its X and Y positions to zero and 52 respectively. Set up the border we used earlier as border number one, set the default border to one, position the view window, click on the eye, and press escape: Good innit?

Well, sadly we've reached the end of this rather happy go lucky tutorial. I haven't answered all your questions, but I can guarantee you that the reference section won't seem half as horrifying now, and you should be able to find your own way through from here. I wish you hours of fun and joy using 3D Construction Kit 2.0. Remember: the current record is six days... and he/she/it/they did a **most excellent** job.

8: THE USER INTERFACE REFERENCE GUIDE

8.1 Conventions

Left click	Click with the left mouse button
Right click	Click with the right button
Cursor	The arrow onscreen that follows the movement of the mouse
Menu Bar (activated with a Right click).	The pull down menu section at the top of the screen
Selecting a menu item	Highlighting a particular menu item and clicking with the left mouse button
Scroll bar	Vertical bar used on the text editor, dialog boxes and file selectors to scroll up or down a long list of text. You can use the scroll bar in several ways. By clicking on the Up or Down arrows, you move the window up or down one line. By clicking in the long scroll bar outside of the little scroll box, you can move the display up or down a page at a time, depending on whether you click above or below the scroll box.By clicking on the scroll box and holding the button down, you can drag the screen up or down by moving the mouse.
Button	Onscreen text box which, when clicked on, will perform whatever its name suggests eg. OK
Icon	Onscreen graphic which, when clicked on, will perform whatever function is attached to it
Text Box	Onscreen text box which, when clicked on, will allow you to enter a number or a string of text
Cross-Hairs	Cross that marks the centre of the screen

8.2 The File Selector

PC+Atari ST

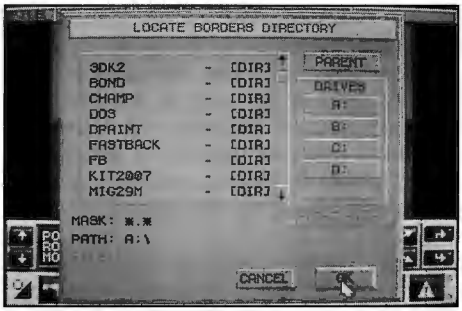


Figure 8.1

Whenever you need to load, save or locate a file on a disk, you will be presented with the 3D Construction Kit 2.0 file selector window (fig. 8.1). At the top of the window is a bar telling you exactly what you are trying to do. Underneath that bar are two sections, on the left is the file selector window which shows all the files and directories in the currently selected path. Attached to the file selector window is a scroll bar allowing you to see all the files in the directory. To the right is a series of buttons. The top button, PARENT, will take you back up one level in a pathname from a sub-directory to a directory. The buttons below the PARENT button, are the available drives and by clicking on one of these you can select the relevant disk. At the bottom of the drive buttons is a button marked FILES+DIRS. Clicking on this toggles between displaying files only or files and sub-directories.

Beneath the file selector window are three text fields : MASK which controls what files are displayed in the file selector window, PATH which contains the current path and FILE which contains the current file.

Finally, at the bottom of the screen is a CANCEL button which will cancel the disk operation you were trying to perform, and an OK button which will return the filename you have selected.

Amiga

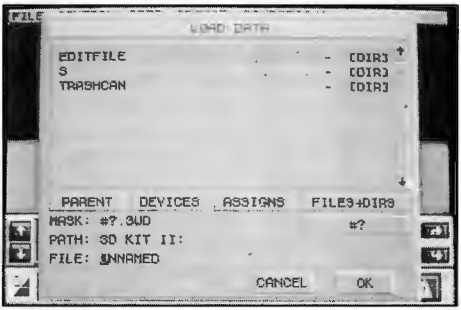


Figure 8.2

Whenever you need to load, save or locate a file on a disk, you will be presented with the 3D Construction Kit 2.0 file selector window (fig. 8.2). At the top of the window is a bar telling you exactly what you are trying to do. Underneath that bar is a file selector window which shows all the files and directories in the currently selected path. Attached to the file selector window is a scroll bar allowing you to see all the files in the directory. Underneath the file selector window is a series of buttons. The first button, PARENT, will take you back up one level in a pathname from a sub-directory to a directory. Next to the PARENT button, are the DEVICES and ASSIGNS buttons. These list are the available physical drives and logical drives, respectively. Clicking on one of these will bring the relevant list in the file selector window, by clicking on one of these you can select the relevant disk. Next to the ASSIGNS button is a button marked FILES+DIRS. Clicking on this toggles between displaying files only or files and sub-directories. Beneath this button is a button marked #?, clicking on this automatically shows you all the files in the directory.

Beneath the file selector window are three text fields : MASK which controls what files are displayed in the file selector window, PATH which contains the current path and FILE which contains the current file.

Finally, at the bottom of the screen is a CANCEL button which will cancel the disk operation you were trying to perform, and an OK button which will return the filename you have selected.

8.3 The Text Editor

Figure 8.3

Built into the 3D Construction Kit 2.0 editor, is a simple text editor for editing conditions. When you go into the editor (usually by either selecting the EDIT section of a menu for one of the conditions, or by clicking on the shortcut icon for EDIT OBJECT CONDITION) you will be presented with a screen not unlike fig. 8.3. At the top of the screen is a bar telling you exactly what you are editing, below that is a large square area where we will be editing text, and to the right of that a bar with a box in which the scroll bar allowing us to go up and down the text we have inputted, at will. Beneath the text entry window is a status line which tells us what type of condition we are editing, its name and number. Beneath that is a row of buttons allowing us to do several useful things :

LOAD	Useful for loading text from disk.
SAVE	Useful for saving any text you have typed in to disk. This is useful for two reasons : firstly you can keep a library of commonly used conditions on disk and just import them when you need them and secondly, you can use the saved text in other applications, or even share it with a friend. The saved condition files are just normal ASCII text files, so you can even edit them outside of 3D Construction Kit 2.0 and import them here.
GOTO	Goto a particular line in the text. Particularly useful if you have a long condition which is longer than the text window can hold.
CLEAR	Will delete all the text from this condition. WARNING : Use this with care, else there will be tears before bedtime!
6X6 (or 8X8)	There are two font (character sizes) in the text editor – 8x8 which is a nice large text for those with poor eyesight or tv`s and a 6x6 font for those with good glasses or monitors. The 6x6, although harder to read does allow you to get more on the screen. The default font size can be set from PREFERENCES (on the GENERAL menu).
CANCEL	For those times when things just aren`t going right, this will reset everything you`ve done so no changes are made.
OK	For those times when its all gone right and you want to keep the changes you`ve made.

When you click on OK after editing a condition, the text is then compiled by the FCL compiler built into the editor. If there is something it doesn`t understand (which is a user friendly way of saying that you`ve typed some rubbish in) then it will display an error message with two buttons OK and CANCEL. Clicking on OK will take you back into the editor with the offending line highlighted, clicking on CANCEL will have the same effect as clicking on it in the editor : all your changes will be lost and you will sent back to the Freescape editor.

8.4 Alert Box

Figure 8.4

From time to time you will do things inside the Freescape environment which will either be incorrect or will cause something irrevocable to happen. When this happens an Alert Box will appear (fig. 8.4). If the function you are trying to do is just impossible, then the system will inform you of this and wait for you to press an OK button to confirm you have read the message. If the function will cause something to happen that can`t be undone then an Alert Box will be displayed, and you will be given the option to reconsider your action.

8.5 Dialogue Box

Figure 8.5

Many of the menu functions, when activated will bring up a dialogue box to allow you to change some parameter (fig. 8.5). Dialogue boxes are similar to Alert Boxes, but they may have any one of three types of buttons :

Function buttons	These do one specific task eg. OK, CANCEL
Radio buttons	These are groups of buttons that are mutually exclusive ie selecting one of them, automatically deselects all the other buttons eg. the WALK/FLY1/FLY2 etc. buttons in the vehicle dialogue box
Toggle buttons	These are buttons that can be toggled between a selected and an unselected state eg. the INV/TAN/WIR etc. buttons in the edit object attributes dialogue box.

As well as buttons there may text boxes to allow the entry of text or numeric data. Text boxes may be edited by first clicking on the box with the mouse. You can then type in your text string. Pressing DEL will delete the character under the cursor and pressing BACKSPACE will delete the character before the cursor. Press RETURN to finish editing. Some text boxes may restrict you to numerical data only.They may also have a scroll box with a sliding bar to allow editing of long lists of data.

8.6 Guide to the Screen

Figure 8.6

When you run 3D editor, you will be presented with a screen that looks like fig. 8.6 (if not, it might be time for that visit to the Opticians). At the top of the screen is the menu bar, to pull down the menu, click in the relevant box with a right click.To select a particular menu item, click on it with a left click. Click outside a menu to remove it. Below the menu is the Freescape window and underneath that is the status bar and user movement controls.

8.6.1 The Status Bar

On the left hand side of the bar underneath the Freescape window is the status window. This window is used so that the system can help you keep track of what`s going on at any time. To the left of the status window are two arrows which allow you to scroll the status window up and down. The various information in the status window falls into four categories:

Figure 8.7

World section (fig. 8.7)

The top row shows the user`s position in the three world axes, the middle row shows the user`s rotation along the three axes, and the bottom row shows the user`s movement mode and height.

Figure 8.8

Object section (fig. 8.8)

The top row shows the current object number and name, the middle row shows the object`s position in the world and the bottom row shows the object`s size.

Area section (fig. 8.9)

The top row shows the current area number and name, the middle row shows the number of objects in the area and the number of facets in the current view. The bottom row holds various status information about the system :

Figure 8.9

M – Amount of time taken to move all the objects and the player through the world and perform collision detection.

P – Amount of time taken to perform the 3D maths.

S – Amount of time taken to sort the objects in display order.

C – Amount of time taken to actually draw the display.

D – Amount of time taken to copy the screen across (only really useful on the PC).

Memory Section PC (fig. 8.10)

The first two rows show the free memory in the system :

Free Memory 1 – The amount of memory used for the world data

Free Memory 2 – The amount of memory used for fonts and brush data

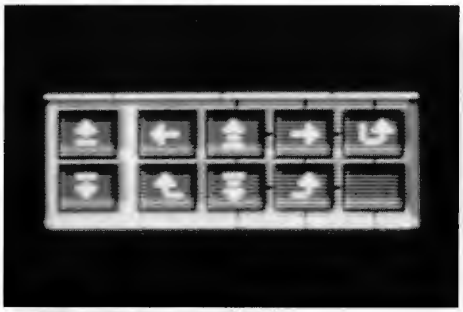
The bottom row shows the total free memory available in the system

Figure 8.10

Amiga

The first row shows the amount of free memory available in the editor, the second line shows the total amount of free memory available in the system.

8.6.2 The Freescape controls (fig. 8.11)



Next to the status window are the Freescape movement icons which allow the user to move about the screen. These are (working left to right) :

Figure 8.11

Top Row :

Cross-Hair : Toggles the centre marking on and off

Turn Left : Rotate the view point left

Move Fwrd : Move the view point forwards

Turn Right : Rotate the view point right

Move Up : Increase the user`s height

Bottom Row :

Flip : Flips your view through 180 degrees

Move Left : Move the viewpoint left

Move Back : Move the viewpoint back

Move Right : Move the viewpoint right

Move Down : Decrease the user`s height

8.6.3 The View Controls

The final section on this bar is the view controls. From left to right these are :

Centre View : Reset the view position to flat and straight ahead

Look Up : Move the horizon down

Look Down : Move the horizon up

Tilt Left : Tilt viewpoint left

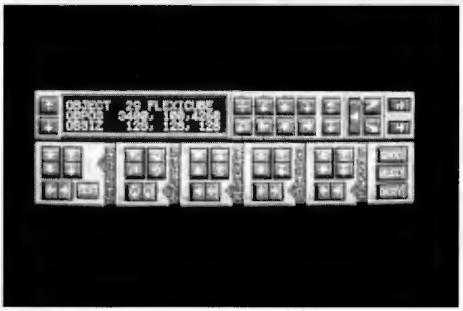
Tilt Right : Tilt viewpoint right

8.6.4 The Shortcut Icons

At the bottom of the screen are the shortcut icons. These are buttons which when clicked on call some of the more commonly used functions. They may be disabled from the menus to increase the Freescape view window size. From left to right these are :

Create Object, Delete Object, Edit Object Condition, Edit Object Attributes, Colour Object, Edit Object, Duplicate Object, Select Object, Test Mode, Reset.

8.6.5 The Object Editor Icons



When you select Object Edit mode, from either the menus or the shortcut icons, you will be presented with a series of icons like fig. 8.12. These are grouped in sections as follows :

Figure 8.12

Point Edit : allows the movement of individual points on an object.

Move point into screen	Move point up	
Move point out of screen	Move point down	
Move point left	Move point right	Select next point

Turn Object : allows the rotation of object about each of the axes

Rotate forwards in Z	Rotate forwards in X
Rotate backwards in Z	Rotate backwards in X
Rotate backwards in Y	Rotate forwards in Y

Shrink Object : allows the selective reduction in size in each axis

Shrink object Z+ axis	Shrink object Y+ axis
Shrink object Z- axis	Shrink object Y- axis
Shrink object X+ axis	Shrink object X+ axis

Stretch Object : allows the selective increase in size in each axis

Stretch object Z+ axis	Stretch object Y+ axis
Stretch object Z- axis	Stretch object Y- axis
Stretch object X+ axis	Stretch object X+ axis

Move Object : moves the whole object in any of the axes

Move in Z+ direction	Move In Y+ direction
Move in Z- direction	Move In Y- direction
Move in X- direction	Move In X+ direction

The final set of icons are system icons. These are :

UNDO	Undo up to 10 previous operations
SELECT	Allows the selection of a new object to EDIT
OK	Finish editing the object

8.6.6 The Colour Panel

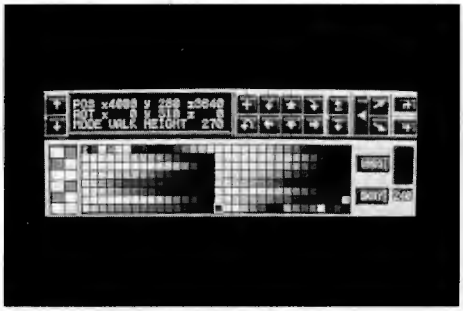


Figure 8.13

When colouring an object you will be presented with a panel like fig. 8.13. This is split into three sections. The first section, on the left, shows the colour of each of the facets of the object your are editing. Clicking on any of these squares will make that colour the currently selected colour. Clicking on any of these squares with the right button will colour that facet in the currently selected colour. The second section is the actual colour palette. This contains all the shades that you can use to colour objects. The first colour (marked with an I) is invisible and any facet with this colour will not be drawn. The currently selected colour is highlighted with a flashing square cursor. By clicking with the left button on any colour in this area you can make that the currently selected colour. Final, on the right hand side are the UNDO button which will reset all the colours to their settings before you entered the colour editor, the OK button which exits with the new colour settings, a large block showing the currently selected colour and it colour number.

In addition to the above method of selected facets, you can click on an object facet (if it is visible in the view window) to select or change colour. Click with the left mouse button to make the facet`s colour the currently selected one, and with the right button to set that facet to the currently selected colour.

8.6.7 The Debugger Screen

Figure 8.14

To enter the debugger click on DEBUG in any of the condition menus. Once you have selected the particular condition you want edit, you will be presented with a panel like fig. 8.14. In the top left hand corner of the panel is the name and number of the condition you are debugging. Next to this are the flags : Z is the zero flag which is used in conditions – a 0 here means the condition is true, M is the mode flag – if in a conditional statement this will show if the program is waiting for another condition and how it will be merged with the current condition state. The M flag can either be A to show that the next condition will be ANDed, O for ORed, N for NOTed or a "-" if it is not active. The final flag, F, is the wait state flag and these can be can be toggled on and off by clicking on them. The two flags are W for WAITing and T for WAITTRIG.

Below the information panel is the command list. This displays three lines of the condition your are debugging. You can scroll up and down your condition by using the up and down arrow icons to the left of the list (with the left mouse button), or by using the up and down arrow keys. The end of the condition is marked by —{END}—. The current PC (program counter) position is marked by a >> symbol in front of the command and you can change its position by using the right mouse button on the scroll icons.

Below the command list is a variable display containing the value of three numeric variables and one string variable. You can modify which variables this panel is looking at by clicking on the variable name : a dialogue box will appear allowing you to enter a new variable number. At the top right of the control panel are three buttons : EXEC which will execute the command at the PC, SKIP which will skip the current command and move the PC to the next command and EXIT to leave the debugger.

9: THE MENUS REFERENCE GUIDE

This section of the reference manual gives details of every option available on the menu bars and the program components altered and alterable by and through these commands. The section is ordered left to right across the menu bar and then top to bottom on each bar.

9.1 The File Menu

This contains various function for interacting with the outside world.

SAVE DATA	Saves an entire world. This includes all areas, objects, conditions and sound data.
LOAD DATA	Loads an entire world. This includes all areas, objects, conditions and sound data. NB This will overwrite any data you currently have stored in the program.
SAVE AREA	Saves the currently selected area. This will include all the objects and conditions associated with that area.
LOAD AREA	Loads the area from disk into the currently selected area.This will include all the objects and conditions associated with that area. NB This will overwrite any data in the current area.
SAVE OBJECT	Saves an object (or group) and its associated condition(s).
LOAD OBJECT	Loads an object or group. A dialogue box will be brought up to allow you to load the object at one of three positions. Either at its saved position, or into the current area at the current viewpoint, or at a position and rotation specified by the user.

SOUND

The sound entry section.

PC

LOAD	Load a sound file in to memory.
INFO	Shows an information box with the currently selected audio driver, the drivers available and the amount of memory consumed by any loaded sound file
REMOVE	Delete the loaded sound file from memory. Memory is returned to the system.

Amiga

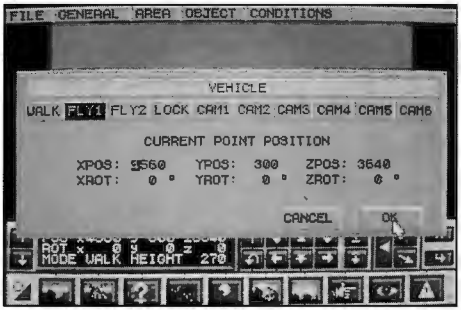
LOAD SAMPLES	Load a sound module in to memory.
EDIT SOUNDS	This brings up a dialogue box that allows you to edit the sounds (see the sound editor section for details).

BORDERS

LOCATE	Allows you to tell 3D Construction Kit 2.0 in which disk folder the borders are stored. NB All borders used within a 3D Construction Kit 2.0 designed world must be stored in the same folder.
ADD	Adds a border from the located disk folder to the borders list.
REMOVE	Removes a border from the border list.
VIEW	Allows viewing of any border on the border list.
EDIT	Allows editing of the border information for any border on the border list. It brings up a dialogue box which allows you to change the border name, whether the current area colours are to be as kept or whether they are to be changed to those in the picture, and whether the border is to be cleared in the area currently occupied by the view window.

CLEAR ALL	Resets 3D Construction Kit 2.0 as if the program had just been loaded. All data is cleared from memory (and should therefore be saved first).
DELETE FILE	Deletes a file from the disk.
SHORTCUTS	Turns the bottom icon bar off thus enlarging the view window. All of these functions are also available through the menus as noted within this guide.
ABOUT	Credits for 3D Construction Kit 2.0
SUPERSCAPE	You played with the home version now use the professional version like the big boys!
QUIT	Leave 3D Construction Kit 2.0 and return control to the operating system.

9.2 The General Menu



The menu with all the user setup stuff.

MODE (fig. 9.1)

Allows the selection of various movement and display modes.

Figure 9.1

VEHICLE

Allows you to set your viewpoint in the game and your mode of movement as well as allowing you to make absolute moves to any point in the area and any rotation. The movement modes are as follows :

WALK is the default mode that only allows ground travel.

FLY1 allows aerial movement but forwards movement always remains parallel to the ground.

FLY2 is like FLY1 but forwards movement is calculated relative to rotation around all of the axes, which means that you move in the direction you are pointing.

LOCK and CAM1-6 These are general purpose view positions which can be set to whatever values you require and later recalled to set you back in the same position.

HIGHLIGHT

Allows control over the way various objects are displayed. When HIGHLIGHT is on the currently selected object will have highlights flashing around its extremities (can you say that legally?) and when EXCLUDE is on, only the currently selected object will be drawn.

LOCK

Asks you to select an object from the object list. The program will then move you so that you are looking directly at the selected object from whichever direction you specify. Your mode will change to LOCK, as if selected from the Vehicle panel.

PREFERENCES (fig. 9.2)

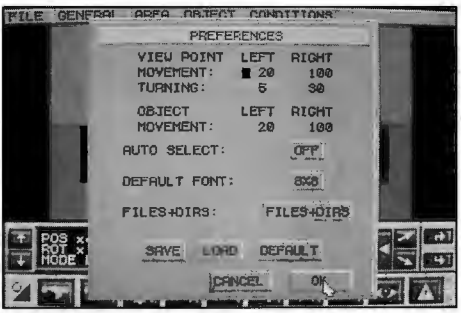


Figure 9.2

Allows you to set 3D Construction Kit 2.0 up as you would like it. If the preferences set on the panel are saved then 3D Construction Kit 2.0 will automatically load with these preferences installed. The editable options on the panel define your movement rate, object movement rate (useful when editing objects), the default font (either 6 x 6 pixels or 8 x 8), and whether auto select is on or off. When auto select is on, the object currently selected on the object list will be automatically used for any object manipulation command. This saves a lot of repeated selections of the same object from the object list. To change the selected object either click on the eighth icon button (a hand with a finger pointing right), or change objects by clicking on them in the usual manner.

DEFAULTS (fig. 9.3)

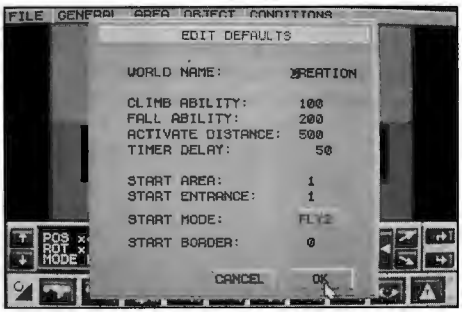


Figure 9.3

Allows you to set the game's initial state / state after a reset. Editable parameters are the name of your world, Climb and Fall abilities ie the maximum distance you can climb and the furthest distance you can fall (nothing actually happens to you if you fall except for setting the maximum distance fallen system variable (V10)). You can also set the furthest distance you can be from an object and still Activate it, the Timer Delay value, the Start Area and Entrance, initial movement Mode, and the first Border that will be used. The Timer Delay is general purpose timer that is cleared every time a reset is performed in the world. When it has passed the value in DEFAULTS (in units of 50th of a second*) then the TIMER? condition in FCL will become true.

* 60th of a second on NTSC Amigas

RESET

Reset various sections of the virtual world.

OBJECT — Resets an object's attributes back to their default state. This will not reset any changes made using the Edit Object option, but if an object has been faded, activated, moved,

etc., during the course of testing then it will be placed as it was when you began the test.

AREA — As reset object but applies to every object in the area rather than one specific object.

WORLD — Resets everything including your position but does not clear any data. This command is also performed by the exclamation mark icon in the shortcuts panel.

CONTROLS (fig. 9.4)

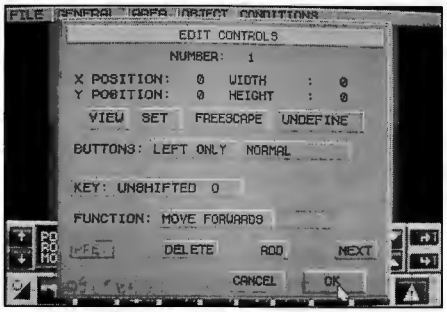


Figure 9.4

Allows you access to the control editor. You can set a control's position on screen either in absolute terms or by visually dragging and sizing a box (using the Set option). The FREESCAPE button will set the icon size and position values to -1, which indicates that the icon definition will be the same as the Freescape view window. The Undefine button will reset the button to the state it was in before you started messing around. You can assign a key on the keyboard as well as a mouse movement to each control. The functions of the keys are defined but extra controls can be specified by procedure number.

INSTRUMENTS

CREATE — Adds a new instrument of a type specified on the panel to the instrument list. Text windows are used for alphanumeric text, numerical instruments for numbers, horizontal and vertical instruments for strength/energy bars, and dials for clock type readings.

EDIT (fig. 9.5)

Figure 9.5

Gives access to the editing panel where instrument parameters may be defined as follows: Name, position and size on screen, source variable for the instrument, range (upper and lower points), foreground and background colour, and font. Allows drag and stretch setting of instrument on border. *Store must be clicked* to finalise edit before changing instrument or leaving the panel, otherwise new information will not be stored.

DELETE — Removes an instrument from memory and erases its definition.

BRUSHES

Allows the use of small square sections of graphics from other graphics programs.

CUT BRUSH (fig. 9.6)

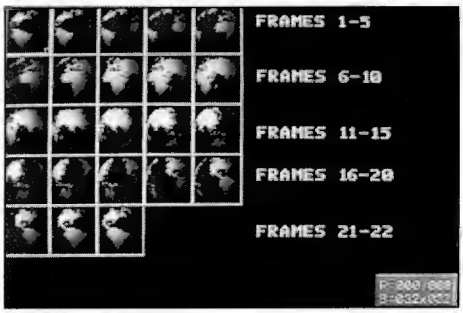


Figure 9.6

Will ask you to select a picture file to load and then display it on the screen. There will be a square dotted drag box which you can move around the screen by clicking on the left button and holding it down whilst you move the mouse. You can adjust the size of the brush by clicking and holding down on the small box in the corner of the larger one. When you are happy with the position click with the right mouse button.

EDIT BRUSH — Using this mode you can preview all the brushes that have been cut in this session. Click on NEXT and PREV to scroll through the brushes.

DELETE BRUSH — Useful for removing any redundant or incorrectly cut brushes.

BRUSH ANIMATIONS

This section of the menu is used to composite brushes together to make animations.

CREATE	Click on this to create a new animation.
EDIT	Clicking on this brings up the edit dialogue box.
DELETE	Remove an animation from memory.

Edit Animation Dialogue Box

This contains several fields (user alterable ones are in bold):

Name :	The name of the animation.
Number :	The animation number
Status :	Enabled or disabled – disabled animations cannot be activated from within FCL
X and Y positions :	The position the animation will appear on screen
Initial Mode:	This is the way the animation will be sequenced. These are :

REPEAT	Continually repeat the animation. When the animation reaches an end then skip back to the beginning if going forward else the end.
BOUNCE	Move backwards and forwards through the animation.
RANDOM	Display frames randomly.
INACTIVE	Do not display (unlike DISABLE this can be over-ridden by FCL).
STOPPED	Do not animate but do display the current frame.
SINGLE	Do animation once in the specified direction.

Direction :	Describes which way the animation will be displayed, either from the first frame to last (FORWARDS) or from the last frame to the first (BACKWARDS).
EDIT	Click on this to edit the brush list
PREVIEW	Click on this to see your animation in action
CANCEL	For those times when it all gets a bit much and you don`t want to keep what you`ve done.
OK	Walt has nothing on you – save that animation for later!

Edit Brush List

This allows you to choose the actual brushes you want to use and what order they are to appear in. Click on ADD to bring up a list of available brushes. Select a brush and click on REMOVE to remove it. Click on CLEAR to clear the whole list (drastic I know but sometimes you just have to do it!).

SET VIEW WINDOW

Allows the setting of the view window either by the direct entry of on screen X and Y coordinates or by a drag and stretch method by clicking on SET. When in SET mode, clicking on the coordinate display will allow you to load in a border for reference.

NB On the PC version, due to hardware limitations, the view window size and position in the X axis can only be a multiple of 8.

EDIT USER FADE (fig. 9.7)

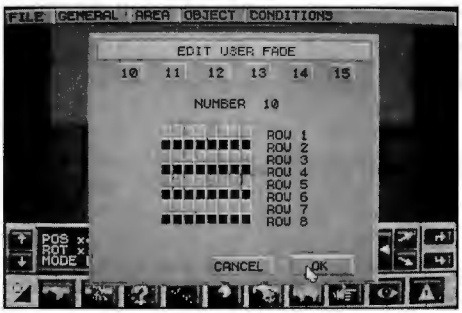


Figure 9.7

This allows the editing of the user fade masks. At the top of the dialogue box are a row of six buttons numbered 10 to 15, these select one of the user fades. Below that is a matrix which is the actual fade mask. A black square in the matrix means a solid pixel will be plotted, an empty square indicates nothing will be plotted. At the bottom are the good old CANCEL (to forget your changes) and OK to save them.

PALETTE *(PC Only)*

Allows the user to change the palette of colours at their disposal. You will be presented with Red Green and Blue sliders to alter the individual components of a colour. Click on SELECT to choose which colour to edit. Click on VIEW to see the result of your colour change (right click to return). Click on CANCEL to forget your changes and OK to save them.

VIDEO

This allows you to record a sequence of actions and play it back using FCL.

CREATE SEQUENCE Places an empty sequence on the viewpath list.

RECORD SEQUENCE	Once the viewpath has been selected, the user is presented with a "VCR" type control panel on which they may record and play back a series of moves within Freescape. The buttons on the video recorder, left to right, represent Stop, Record, Rewind to start, Play, and Mark end point (all information from the current position onwards is erased). The information displayed is time and moves as both complete totals and to the current position. The name and number of the viewpath are also shown.
EDIT SEQUENCE	Allows editing of the selected viewpaths parameters. Allows the start point of playback to be defined or for the viewpath to be followed relative to the player's position. Playback modes are Realtime (exactly as entered) and Normal (where the moves are made but any pauses between them are removed) and are toggled on the panel. The viewpath can be made to repeat a user specified number of times (the repeat count) and should a defined start position be desired for the viewpath it can be stipulated on this panel as either an area/entrance specific or by both position and rotation co-ordinates.
DELETE SEQUENCE	Removes a sequence from the viewpath list.
TEST	Puts the program into test mode (but does not reset the game) allowing the user to interact and test their controls, instruments, borders, and general gameplay. To perform a reset (and thus load the border) the user must press the escape key. This command is functionally identical to the ninth (eye) icon on the Shortcuts panel.

9.3 The AREA Menu

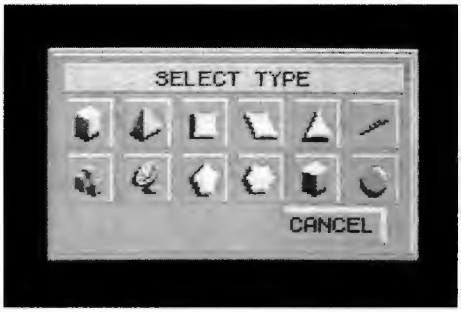
Contains various functions to manipulate Areas.

CREATE AREA	Adds a new "unfurnished" area to the area list.
EDIT AREA	Allows alteration of a selected area's basic parameters: its name and whether it has a visible horizon. An area's scale is also defined here and the default border to use. NB Although each area has its own default border, the area defined as the START area will use the BORDER defined as the start border in DEFAULTS.
DELETE AREA	Deletes the specified area from the area list.
GOTO AREA	Puts the selected area in view and allows its objects, conditions, etc. to be altered.
AREA COLOURS *(Amiga only)*	Allows the editing of the 16 colours used for each area.
CREATE ENTRANCE	Creates an entrance at the user's current position and rotation and adds it to the end of the entrance list.
EDIT ENTRANCE	Gives access to the entrance panel. The list can be moved through with the "next" and "prev" buttons and the position of the entrance can be set either through entry of absolutes for position and rotation, or by setting the user's current position as the entrance. Other buttons allow the user to go to the entrance and also to view from the entrance (the user's position and rotation remain unchanged, and the display returns to current as soon as the mouse button is released). An entrance's name can also be changed here.
DELETE ENTRANCE	Removes an entrance from the entrance list.
GOTO ENTRANCE	Moves the player to the selected entrance.

9.4 The OBJECT Menu

Everything you need to mess around with objects.

CREATE OBJECT (fig. 9.8)



Takes a primitive (basic object shape) from the available options on the panel and adds it to the object list, placing it in the players view. Groups (three small cubes) and Sensors (satellite dishes) can also be created from this menu.

Figure 9.8

EDIT OBJECT

Allows manipulation of the physical conditions of a selected object. The corner points of most objects can be individually moved, all objects can be turned, shrunk and stretched, and moved around the area. It is through this command, which is identical in function to the sixth icon button on the Shortcuts panel (box with red extensions), that the visual architecture of the world is performed.

DELETE OBJECT

Deletes an Object or group from the Object List.

SELECT OBJECT

Selects an object from the object list. This function, also performed by the eighth (pointing finger) icon button is used in conjunction with auto select (see the PREFERENCES menu entry).

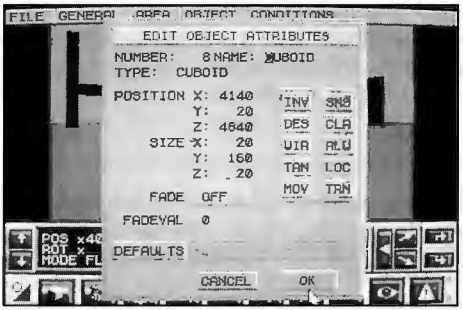
COPY OBJECT

After the desired object or group has been selected the player is asked to position the duplicate either beside one of the faces of the subject, or in absolute terms of position co-ordinate. This function is also performed by the seventh icon on the Shortcuts panel (two boxes and a red arrow) .

ATTRIBUTES

This command functions in the same way as the fourth (tick and cross) icon button. Performs differently depending on whether the selected item from the object list is an object or a group.

For Objects: (fig. 9.9)



Allows exact positioning and sizing, fade control, and control over the object's attributes through toggle button. The toggle buttons are representative of the following object capabilities:

Figure 9.9

INV : Whether an object is visible (off) or invisible.

DES : Whether an object is destroyed or not (off).

WIR : Whether an object is represented as solid (off) or wireframe.

TAN : Whether the player can pass through the object (on) or not.

MOV : Whether an object is permanently static (off) or movable ie whether it can be included in an ANIMATOR list.

SNS : Whether the object is a sensor (on) or not.

CLR : Whether the object's colour is fixed (off) or changeable.

ALW : Whether the object condition executes only on collision / activation / shooting (off) or whether the condition always executes.

LOC : Whether the object will be animated if the group of which it is a part is animated (off) or not.

TRN : Whether the object is a Transporter (on) or not.

If the selected object is a transporter or a sensor then access is given to two other panels as follows:

SENSOR: Direction of sense can be set as well as any sound to be activated on triggering the sensor. Any procedure designed to be activated is named here, as is the sensor's type (shooting or detection oriented), range, and speed of response.

TRANSPORTER: The area and entrance to which the player will be transported are set on this panel.

DEFAULTS: Set the attributes that will be applied to the object after a reset is done. The ones affected are INV, WIR, DES, TAN, object position and FADEVAL.

For groups:

Access is given to a panel that allows the group's name to be changed, and also allows objects to be added to the group by selecting them from the object list after clicking on add, or removing them by highlighting the relevant object in the group's object list (left side of the panel) before pressing remove. Identical to the menu option EDIT GROUP.

COLOUR — This option provides the same facilities as the fifth icon on the Shortcuts panel (three intersecting coloured circles). For more information see the User Interface section.

EDIT GROUP (fig. 9.10)

After selecting a group, access is given to a panel that allows the group's name to be changed, and also allows objects to be added to the group by selecting them from the object list after clicking on add, or removing them by highlighting the relevant object in the group's object list (left side of the panel) before pressing remove. Identical to selecting a group from the object list when in OBJECT ATTRIBUTES.

Figure 9.10

9.5 The CONDITIONS Menu

This is the menu to come to when you want to mess about with FCL conditions (and why not!).

GENERAL CONDITIONS

These are conditions which execute anywhere in the world.

EDIT	Enter those General conditions here (see the User Interface section on the Text Editor for more details).
CREATE	Creates a new blank general condition which is added to the General condition list.
DELETE	Removes a General condition from the list.
INFO	Allows access to a panel of basic parameters for the General condition. The condition's name may be changed and it may be enabled or disabled here. This panel also allows the user to decide whether the condition continually executes, waits for a frame update between executions or is not executed until it is triggered by another Condition / Procedure / Animator.
DEBUG	Got those problems then this is your man. (See the the User Interface for details on the Debugger).

LOCAL CONDITIONS

These are conditions that are specific to the area and operate when the player is in the area. The area used is the one currently in view.

EDIT

CREATE

DELETE

INFO

DEBUG

All of these commands work in the same way as those listed in General Conditions but apply to Local Conditions (obviously).

OBJECT CONDITIONS

These are conditions that are run when an object is shot, collided with, or activated. They can be set to run always by clicking on ALW on the object attributes screen.

EDIT	Works identically to the editor for General Conditions but applies to objects. This command is identical to clicking on the third (question mark) icon button.
INFO	This works similarly to the Info command for General conditions except the process flags are available for setting and clearing. Here's what they mean:
DEBUG	Identical to Debug on General Conditions but applies to Object Conditions.
ANIMATORS	These are groups of FCL commands that move objects and groups around the area.

EDIT

CREATE

DELETE

INFO

DEBUG

All of these commands work in the same way as those listed in General Conditions but apply to Animators.

INITIAL CONDITIONS

These are Conditions which execute when the program first starts, and when the game is reset.

EDIT

CREATE

DELETE

INFO

DEBUG

All of these commands work in the same way as those listed in General Conditions but apply to Initial Conditions.

PROCEDURES

These are pieces of FCL code that can be called from within the other forms of conditions. They can also be assigned to controls, or triggered by sensors.

EDIT

CREATE

DELETE

INFO

DEBUG

All of these commands work in the same way as those listed in General Conditions but apply to Procedures.

CONDITION ENABLES

This allows you to turn any or all of the conditions on or off both in the editor, and in game testing. Each type of condition is listed and its status in both the game and the editor may be toggled.

ENABLE ALL

Sets all Condition Enables to On in both the editor and the play tester.

DISABLE ALL

Sets all Condition Enables to Off in both the editor and the play tester.

NB Enable/Disable all do not affect the individual condition enables but are only an overall toggle.

10: Introduction to the Freescape Command Language (FCL)

By now you should be reasonably happy with creating quite complex objects by starting with the simple primitives and using the editor to bend, stretch, re-colour and combine them. Maybe you are a little uneasy about building a utopia starting with two cubes and a straight line or, on the other hand, maybe you get a buzz out of stuff like that! Either way, you should be capable of creating something and moving round the world to view it from different angles. The problem at this point is that it does not do an awful lot, other than sitting there and looking terribly pretty (or wonderfully ugly, depending on your sense of aesthetics). What we need to be able to do is to make objects interact with each other and the user using the Freescape Command Language (FCL).

What is FCL?

Imagine that the first thing you do in the morning is make a list of the chores you have to do that day (that`s funny : the first thing most people do in the morning is wake up). It might look something like this :

. Make breakfast

. Take kids to school

. Do the shopping

. Pick the car up from the garage

. Buy a birthday present for Bob

This list of instructions could then be broken down even further. For instance, "Make breakfast" could be broken down into:

. Get milk from fridge

. Pour milk into glass

. Put bread into toaster

etc.

Eventually, you will end up with a list of tasks that can not be broken down any further. OK, I hear you say, this is great for my cholesterol level, but what has it got to do with FCL. Well, FCL is a set of english-like commands, that the computer understands, which perform individual tasks. These commands are analogous with our simple breakfast tasks. These can then be combined in a list, like the one for making breakfast, to perform more complex tasks. These lists, or programs, are known as **Conditions** in FCL.

Starting to program

Before we start programming, we need to do a bit of setting up. Select the CLEAR ALL option from the FILE menu and click on YES when the warning box comes up. Now click on the create object icon and select a cube. This should put a nice cube in the middle of your view (if not, then its probably time to go and have a lie down). From now on we will call this our standard setup. We are now ready to write our first piece of FCL! Go up to the CONDITIONS menu and select EDIT from the OBJECT CONDITIONS section. When the object select menu comes up, select object number 2 (CUBE). You should now be presented with a large grey box with the words "EDIT OBJECT CONDITIONS" at the top. This is the text editor used to edit all Conditions (for more information on the text editor see Appendix 2). Type in the following FCL code as it appears here (press the RETURN key at the end of each line) and click on OK.

```
INVIS(2)
```

If a warning box with an error comes up go back to the editor and check that the line that is highlighted is correct. Once you have correctly entered the program you should be back at the main edit screen. Move the mouse cursor over the cube you created earlier and click either mouse button. All being well, the cube should disappear. If it does, congratulations, you've just completed your first FCL program (you may have to move closer to the cube to get it to disappear with the right mouse button). If it does not disappear, go back to the object condition editor and check that the code is exactly as above.

"That's all very well," I hear you scream in frustration , "but what's going on?". Well INVIS is a FCL command that makes an object invisible. The 2 in brackets tells the INVIS command that object number we want to make invisible is our cube, and is known as the command parameter. Most (but not all) commands have parameters, and parameters are always in brackets. Now, if you remember that far back, we said that object conditions are only normally executed when the object is "interacted" with. So, when we press either mouse button over our cube, the condition for that object which we have just written gets executed, making the cube invisible.

Before we go any further there are some basic principles we need to know.

1) Conventions

Program: any small piece of a FCL condition.

PC or program counter: this marks the next line to be executed in your FCL condition.

2) Variables and constants

The world we live in is littered with variables and constants (unless you live in California, which occupies a different universe to everywhere else). Simply put variables are quantities that change and constants are quantities that don`t. So, for instance, the speed of light is a constant but the speed of a car is a variable. In FCL, there two sorts of constants and three sorts of variables.

i) Numeric constants: As the name implies, these are simply numbers. They can be specified in one of two ways, either as normal decimal numbers that everybody knows and loves, such as 14 and -567, or as hexadecimal numbers. Hexadecimal (or hex) numbers are calculated in base 16 rather than the base 10 we are used to and have a $ symbol in front of them to distinguish them from normal numbers, for example $1456, or $FFFFAEF (I know it`s hard to believe but this really is a number. Hex numbers are only of use to the great unshaven of the world who like to stay up to 6:00 am with their eyes glued to their monitor. In other words only programmers use them).

ii) String constants. These are used for normal text and are always surrounded by double quotes.

eg

"THIS IS A STRING CONSTANT"

"and so is this"

but this isn`t – just making sure you are paying attention!

It is important to note the difference between a numeric constant and a string constant containing a number. If you need to use a number in a calculation then it is stored as a number, however if you want to include it in a string it must be surrounded by quotes.

eg

1234 is a number, "1234" is a string

NB You`ll be grateful to know that no programmer has yet come up with a new text system that has 35 letters instead of 26 and that works in seven dimensions!

iii) Numeric variables. Quite often when you are programming, you will find that you need to keep track of a value that may be changing, such as speed. Unfortunately, a constant is no good if the value is constantly changing. You can change the value of a variable whenever you need to. Numeric variables always start with a V and finish with a number between 0 and 511. The variables V0 to V29 are known as system variables and are covered in detail later in this manual.

iv) String variables. In the way that numeric variables store numeric values that change, string variables store text strings that can change. String variables start with an S and finish with a number between 0 and 31.

v) Arrays are like numeric variables but each one is made up of one or more elements, each of which can contain a number. If you imagine a variable as a bucket, then an array is a row of buckets. They always start with an A and end with a number between 0 and 15. Before you use an array you have to tell 3D Construction Kit 2.0 how big the array is going to be, this is know as dimensioning the array. Once this is done then each element of the array is accessed by putting a number in square brackets after the variable name, this is known as the subscript.

eg

DEFARRAY(A1,10)	Give array A1 10 elements
SETVAR(4,A1[5])	Set the 5th element of the array to 4

NB Array subscripts start at 0 rather than 1, so in our previous example the first element of the array A1 will be 0 and the last element will be 9.

3) FCL Commands

There are many different types of commands in FCL but they can be roughly divided into eight types :

i) Variable modifiers. These are commands that are used for setting and modifying the values of variables. For example, ADDVAR which adds two variables together.

ii) System commands. These are commands which cause special things to happen in the system, such as DELAY, which will make the system pause for a given amount of time.

iii) I/O (Input-Output) Commands. These are commands that allow FCL to interact with the outside world. They fall into three categories:

Graphics commands – allows the user to read and write to the screen to draw circles, print text etc.

File commands – these allow FCL to access the disk filing system

System I/O – allow FCL to get various data from the computer, such as date, time, key presses etc.

iv) Object commands. These commands allow the user to modify objects from within a FCL condition. As well as commands to stretch, shrink, rotate etc, there are commands to change the object's attributes, such as making it visible and invisible.

v) Loop commands. These are commands that are used to repeat sections of FCL code. To go back to our breakfast analogy, we might have a procedure called make orange juice that looks something like this :

. Put Orange in juice extractor

. Extract juice

. Pour into glass

This is great if you only want a small quantity of juice, but if we want more than that then we need to repeat the action.

eg

. For each orange

. Put Orange in juice extractor

. Extract juice

. Pour into glass

. Repeat until there are no more oranges left

In FCL there are two types of loop : the LOOP ..... AGAIN loop and the FOR .... NEXT loop (thus winning the award for the most amount of loops in one sentence).

LOOP....AGAIN

The command LOOP is placed at the start of the piece of code you want to repeat one or more times and has one parameter, which can either be a number or a variable, which tells FCL how many times you want the code to loop. The end of the piece of code is marked by the command AGAIN.

eg

```
SETVAR(0,V30)          ; Make V30 zero
LOOP(5)                ; Repeat this code 5 times
  ADDVAR(1,V30)        ; add 1 to V30
AGAIN                  ; End of repeat loop
                       ; V30 will now equal 5
```

It is important to remember that the LOOP command only checks the loop value the first time it is invoked, so if you use a variable as the loop counter, that variable can be modified within the loop with no ill effects at all.

eg

```
SETVAR(10,V30)          ; Make V30 ten
LOOP(V30)               ; Repeat this code V30 times (ie 10)
  ADDVAR(1,V30)         ; add 1 to V30
AGAIN                   ; End of repeat loop
                        ; V30 will now equal 20
```

FOR....NEXT

As with LOOP...AGAIN, this allows the execution of a piece of FCL one or more times but it differs in the way that it calculates the number of times to execute the code. FOR takes four parameters :

FOR(P1,P2,P3,P4)

P1 is a variable that will be used as the loop counter

P2 is the initial value that variable will be given

P3 is the value at which we want the loop to stop

P4 is an optional value saying how much we want to add each loop

When the FOR is executed, P1 is set to P2. The code is then executed until NEXT is reached. At this point P4 (or 1 if P4 was not specified) is added to P1 and if P1 is not equal to P3 then the program counter jumps back to the first line after the FOR (I hope you're following this – I may be asking questions later). So, for instance the line

FOR(V30,0,10)

would execute the code 10 times, with V30 starting at 0 and ending at 10. It is important to note that when V30 is 10 the code is not actually executed, because the FOR....NEXT only works UNTIL P1 is equal to P3. It is also important to remember that because the variable specified by P1 is checked every time around the loop, changing the value within the loop could have odd effects.

vi) Conditional commands.

Time for some practical work. If you think that I am mad then I want you to stand on one leg and shout else continuing reading. OK, you can stop shouting now, I SAID YOU CAN STOP .... thank you. "Ah," I hear you say, "the pressures finally got to him and he has gone completely mad", but you are wrong (maybe). This was just a sneaky way of introducing the concept of conditional commands. These are commands that tell the FCL system to only execute a particular piece of program if certain specified conditions are true. For instance, I said earlier that, normally, object conditions are only executed when one of three interactions with the object take place ie left mouse button with cursor over object (known as Shooting), right mouse button with cursor over object

(known as Activating), or colliding with the player or another object (surprisingly enough, known as Colliding). It would be useful when the condition is executed, to know why it was executed and to react accordingly. This can be done using conditional statements.

Time for an example methinks. Click on RESET to get your cube back. Go back to the edit object condition screen for our cube, click on CLEAR and click on YES when the warning box comes up. Type in the following text and click on OK (again, if a compiler error box comes up, go back to the editor and make sure that you have entered it exactly).

```
IF SHOT?
THEN
INVIS(2)
ENDIF
```

Let`s go through the code line by line to see what's happening :

IF SHOT?

The first word, IF, marks the beginning of the conditional block. The SHOT? is a command which checks to see if the object condition was executed because the cube was shot. There are a series of indicators, which are used whenever an object is interacted with, these are known as the Object Flags. These can be checked, as we do here with the shot flag, in the condition to find out what happened.

THEN

This marks the beginning of the program to be executed. This is useful because, as you'll see later, its possible to chain conditions together to make quite complex checks.

INVIS(2)

As before, this makes the cube disappear. The difference now, is that it will only happen when the object is shot.

ENDIF

This tells FCL where the end of the conditional section is so that it knows how much program to skip over if the condition is not satisfied.

Whilst it is very useful to be able to only execute pieces of program if a certain situation occurs, it would be even more useful to execute one piece of program if the condition is satisfied (known as the condition been TRUE), or another piece of code if the condition is not satisfied. To go back to breakfast again, we could say :

. If oranges in fridge

. then

. make orange juice

. else

. jump out of window

(This may seem a little extreme, but have you ever woke up on Sunday morning with no O.J. in the fridge?)

Go back into the object condition editor and modify the program so it looks like this:

```
IF SHOT?
THEN
INVIS(2)
ELSE
 UMOVE (10,0,0)
ENDIF
```

Again, try clicking on the cube with the left mouse button. As before, the object disappears. Click on RESET to get the cube back. Now click on the cube with the right mouse button – see it trying to run away! The difference now is that we have put an ELSE section in. When FCL reaches an ELSE, it checks the condition that was specified in the previous IF statement for been in the OPPOSITE state. In our example above, this means that the program between the THEN and the ELSE will be executed if the object was shot, and the program between the ELSE and the ENDIF will be executed if the object was NOT shot (in the case of our cube, this would mean we either Activated it or Collided with it).

Another useful feature of conditions is the ability to chain them together. Back to breakfast again (is this making you hungry yet?)

. If there are oranges in the fridge AND I am thirsty

. then

. make some O.J.

. else

. go back to bed

The important thing to note here is the use of the word AND to give what is known as a compound condition. This means that I will only make orange juice if we have some oranges and I`m thirsty, any other combinations, such as I am thirsty but I have not got any oranges will result in my going back to bed (and quite rightly too, in my opinion). The same effect can be achieved in FCL, albeit without the oranges, by the use of the AND keyword.

eg.

```
IF SHOT? AND SOLID?(2)
THEN
  MAKEWIRE(2)
ELSE
  MAKESOLID(2)
ENDIF
```

Try shooting the object now. You should find that it switches between wireframe mode and solid mode. What`s happening is that the MAKEWIRE section is only executed if the object has been shot AND it is currently solid. In all other cases, such as when the object was shot when it was wireframe, will cause the MAKESOLID command to be executed.

As well as being able to chain conditionals using the AND command, it is also possible to chain them using the OR command. This is similar to the AND command but in this case the condition is TRUE if either or both of the conditions are true. So, if the first line of our program was changed to

```
IF SHOT? OR ACTIVATED?
```

then the MAKEWIRE line would be executed whenever either mouse button was pressed over the object.

It also possible to reverse or invert the sense of a condition by adding the command not in front of it, so, therefore, the line

```
IF NOT SHOT?
```

would be true if the object was not shot.

vii) Program commands. These are special commands that modify the way the FCL condition runs. For instance, the END command will force FCL to stop executing this condition.

viii) Animator commands. As well as the ability to move individual objects using their FCL conditions, 3D Construction Kit 2.0 has the ability to chain objects and groups together to perform global movement commands on them. These are covered in the advanced section.

4) Debugging, or what to do when it all starts going horribly wrong!

In the course of your adventures in programming, you will discover, from time to time, that the conditions you have written are not reacting the way you expect them to. Of course, you will blame the computer, you will jump and shout in frustration, you may even feel the need to pull a very big gun out and ask the computer "Do you fell lucky, punk?". Unfortunately, no matter how hard you try, you really cannot lay the blame at the feet of you computer (not least of because it does not have any). You see, computers are very pedantic, they only ever do EXACTLY what you tell them to do. The trouble is that what you THINK you told the computer to do, and what the computer KNOWS you told it to do, can quite often be two different things entirely. When this happens, your program has a BUG. There are several strategies for removing bugs from programs but, quite often, the simplest is to actually plan exactly what you want to do. Although the temptation is to just sit down at the keyboard and start writing, having at least a sketch plan of what you want to achieve is a great help. The next step is to look at what you've actually written and see if it really matches what you wanted to write. If this fails, then the next step is to try to trace what is happening. To do this we can use the debugger built into FCL. For information see the reference section on DEBUG. If all else fails then it often helps to sit down, have a cup of coffee and forget what you are trying to do for a bit – its often a lot easier to find a problem on a rested, clear head.

11: Freescape Command Language Reference Guide

11.1 Introduction

FCL is a procedural based language that is integrated within the Freescape environment, allowing the user to control and animate the "Virtual World". Programs written in FCL are known as CONDITIONS and can be used in six different contexts :

1) General Conditions. These are called every frame and can be used to perform general purpose functions such as a countdown timer.

2) Local Conditions. These are similar to General conditions except they are only called when the user is in the area that they are attached to.

3) Object Conditions. These are only usually activated when some kind of interaction with the object occurs ie when the user shoots it (left mouse button), when the user activates it (right mouse button) or when the object collides (either with the user or another object). However, it is possible to have object conditions executed every frame by setting the ALW bit (see the Edit Object Attributes section for more details).

4) Animators. Built in to 3D Construction Kit 2.0 is the ability to join objects together and "animate" them. These pieces of animation are controlled by chunks of FCL called Animators.

5) Initial Conditions. These are only run after a reset and are used to do any one-off setting up of variables etc.

6) Procedures. These are useful general purpose chunks of code that you may want to use in more than one place. A good example would be a routine to display the score in a game every time it changes.

FCL is a compiled language that has error checking at both compile time and runtime.

11.2 Constants

A constant, in FCL, represents any absolute value stored in program, which never changes. There are two forms of constants in FCL : numeric and strings (text) constants.

i) Numeric constants. These are stored as signed 32 bit numbers and have the range -2147483646 to +2147483645 (the sign is optional when positive numbers). They can be entered in one of two ways, either as straight forward decimal or as hexadecimal (base 16) with a leading $.

ii) String constants. These are any arbitrary text string surrounded by double quotes (").

11.3 Variables

Variables in FCL are used to hold any quantity that may change. There are three sorts of variable :

i) Numeric variables. There are 512 numeric variables in FCL, numbered V0-V511. The first 30 variables, V0-V29 are known as the system variables and are outlined below, in addition variable V255 is a the only variable not to reset to zero when a reset is performed, and as such could be useful for holding a running total, such as a hi-score.

Name	Status	Function
V0	R/W	Viewpoint X position
V1	R/W	Viewpoint Y position
V2	R/W	Viewpoint Z position
V3	R/W	Viewpoint X rotation
V4	R/W	Viewpoint Y rotation
V5	R/W	Viewpoint Z rotation
V6	R	Current Vehicle type
V7	R	Current height (only valid in WALK mode)
V8	R	Current Area number
V9	R/W	Number of last Area visited
V10	R/W	Distance fallen above max ability
V11	R/W	Number of times shot
V12	R/W	Number of times crushed
V13	R/W	Number of last SENSOR to find you (DETECT mode only)
V14	R/W	Number of times SENSED (DETECT mode only)
V15	R/W	ASCII code of last key pressed
V16	R/W	Button status of last press (1- Left, 2 – Right, 3 – Both)
V17	R/W	Mouse X position at last press
V18	R/W	Mouse Y position at last press
V19	R/W	Frame counter for accurate timing
V20	R/W	Player fire control (note 1)
V21	R/W	Number of shots fired
V22	R/W	Scaled Activate distance (Default Activate distance * Current Scale)
V23	R/W	Ground colour + 256*Sky colour
V24	R	Current mouse button state (1- Left, 2 – Right, 3 – Both)
V25	R	Current Mouse X
V26	R	Current Mouse Y
V27	R	Non-cleared mouse button flag (note 2)
V28	R/W	Current area scale (0-31)
V29	R/W	Reserved – do not use

Variables marked as R/W can be read and written to (although it may not make sense to modify some them), those marked as R should only be read and not modified by the user.

Note 1 : Altering this variable allows control of the way the system reacts when the user presses the left mouse button. Putting a 0 here will disable the players ability to shoot completely. A 1 will enable shooting. Adding a 2 will draw lines from the edge of the screen to the firing point, and adding a 4 enables a firing sound. Finally, adding an 8 enables rapid fire, such that a continuous series of shots will be sent as long as the player has the button down. The default value is 15, ie rapid firing with lines.

Note 2: This is a store of the mouse button flags. It is set when either button is pressed but it is not reset when they are released.

In addition to the above, there is a special variable called ME which is only valid in object conditions and which is set to the object number of the current object.

ii) String variables. Just like string constants, these are used to hold any little bits of text you might need. There are 32 string variables in FCL number S0-S31. Note that unlike numeric variables, string variables do not have a set amount of space allocated for them, individually. The string buffer is 1024 bytes long and this has to hold all the strings in the program, both variable and constant. In addition to this, each individual string can only be a maximum of 250 characters long.

iii) Arrays. Arrays in FCL are just like having a chain of numeric variables with the same name. There are 16 arrays in FCL numbered A0-A15. As with strings, all the arrays in your world are lumped into one big memory pool and thus there is a limited capacity. There are 256 longwords allocated for arrays. To use arrays, you first dimension them using DEFARRAY

eg.

```
DEFARRAY(A1,10)
```

Each individual element of the array can then be accessed by placing an index number in square brackets after the array name.

eg.

```
SETVAR(7,A1[6])
```

will set the 6 element of A1 to 7.

NB Array subscripts go from 0, not 1. So, in our previous example, the first element of A1 is A1[0] and the last element is A1[9].

11.4 Animators

Built in to FCL, is the ability to join objects together and perform movement and animation on them as if they were one object. Before an object can be animated it must be marked as moveable. This is done by setting the MOV flag in the EDIT OBJECT ATTRIBUTES dialogue box. Any attempt to animate an object that is not marked as moveable will have no effect. It is important to note that the grouping of objects together to animate them is entirely separate from the concept of GROUPS within the editor, which are essentially a static form for editing purposes. To animate a group, each individual object must be made moveable. Animator Conditions (AC) are only executed when signalled from another FCL condition using the STARTANIM command. All commands in the AC are executed until the end of list is reached or until a redraw request is issued. If a redraw is issued then execution of AC is stopped and the current program counter is saved. At the next frame, execution is resumed from that point. If the end of the list is reached then the AC is marked as STOPPED and can only be restarted with another STARTANIM.

note 1: Although an AC may be initiated from conditions running in any area, it will not actually start until the user is actually in the area associated with that AC.

note 2: Although most FCL commands are useable anywhere, there are certain commands, such as INCLUDE, which are only accessible from with Animator Conditions.

note 3: It is important to note that objects grouped together in an Animator act as one big object for collision purposes. For instance, imagine that you have two small spheres, separated by gap, that are included in an Animator, and a narrow cube between them. It would not be possible to move the two spheres, such that the cube passes between them, without them colliding with the cube. This is because FCL treats the Animator as one large cube that encloses all the objects within the Animator. To avoid this you would have to handle each sphere with a separate Animator.

12: FCL COMMAND REFERENCE GUIDE

12.1 Format of Guide

Each instruction is listed alphabetically with the following layout :

Mnemonic: 1 **Shorthand:** 2

Type: 3

Parameters: 4

Description: 5

Example: 6

Flags: 7

Notes: 8

See Also: 9

1 Mnemonic field. This gives the name of the function along with the parameter count (if any).

2 Shorthand field. Some instructions in FCL shortened versions which can be typed in to the editor to speed up text entry. The next time you edit the condition these will be expanded to their full form.

3 Type field. This gives the function type that function is classified under. These are :

Variable Modifier :	Any function which alters the value of a variable.
Condition Statement :	Any function that is used as part as a conditional section.
System Modifier :	Any function that alters the way a part of the Freescape system works.
Loop Command :	Any function used to repeat a section of code.
Execution Modifier :	Any function that alters the execution of a FCL condition.
Graphics Command :	Used for display various things onscreen.
IO Command :	Allows access to the external resources of the computer system you are running 3D Construction Kit 2.0 on.
Object Command :	Special purpose function for altering an object's characteristics.
Animator Command :	Special purpose routines which are only valid inside Animator conditions.

4 Parameters field. Describes the parameters required by this routine and what their type should be.

5 Description. What the function actually does.

6 Example. A small piece of FCL to demonstrate the function in action and a short description of what would happen if the code was executed.

7 Flags. Describes what state the flags are left in after this function has run. If it is blank then the flags are unaffected. *

8 Notes. Any warnings or further information you should be aware of.

9 See Also. Other FCL commands associated with this routine.

* Certain functions, notably most of the variable modifier commands, affect the flags. This means that they can be used as part of a condition field:

```
IF ADVAR(4,V30)
THEN
  INVIS(4)
ENDIF
```

This would make object 4 invisible IF the result of adding 4 to variable V30 was 0.

Mnemonic: ABS(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Numeric variable or constant

P2 – Numeric variable

Description: Set P2 to the absolute (ie positive value) of P1

Example: ABS(-10,V30) ; V30 is now set to 10

Flags: True if P2 is zero

Notes:

See Also:

12.2 Alphabetical listing of FCL functions

Mnemonic: ACTIVATED? **Shorthand:** ACT?

Type: Condition Statement

Parameters:

Description: Check to see if the current object was ACTIVATED (ie clicked on with the right mouse button)

Example:

```
IF ACTIVATED?
THEN
  INVIS(2)
ENDIF
```

Will set object two invisible if current object activated

Flags: TRUE if activated

Notes: Only applicable to object conditions

See Also: COLLIDED?, SHOT?

Mnemonic: ACTIVERANGE(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – Numeric variable or constant representing range

Description: Set the players reach to P1, overriding the default setting.

Example: ACTIVERANGE(300)

Set the players reach to 300 units

Flags:

Notes: Overrides default setting

Mnemonic: ADDSTR(P1,P2) **Shorthand:** ADDS

Type: Variable Modifier

Parameters: P1 – String variable or constant

P2 – String variable

Description: Appends the string P1 to string P2 and alters the allocated space accordingly
string

Example: SETSTR("Test",S1)

ADDSTR(" String",S2

PSTRING(S2,0,0)

"Test String" will be printed at location 0,0 on the screen

Flags:

Notes:

See Also: SETSTR, CLEARSTR

Mnemonic: ADDVAR(P1,P2) **Shorthand:** ADDV

Type: Variable Modifier

Parameters: P1 – numeric variable or constant

P2 – numeric variable

Description: Add variable P1 to variable P2

Example: SETVAR(5,V30)

ADDVAR(6,V30)

V30 will now equal 11

Flags:

Notes:

See Also: SETVAR, SUBVAR, CLEARVAR

Mnemonic: AGAIN **Shorthand:**

Type: Loop Command

Parameters:

Description: Decrement the repeat count and if greater than zero return to the relevant LOOP

Example: SETVAR(0,V30)

LOOP(10)

ADDVAR(1,V30)

AGAIN

V30 will be 10 at this point

Flags: TRUE if result is zero

Notes:

See Also: LOOP

Mnemonic: AND **Shorthand:** &&

Type: Condition Statement

Parameters:

Description: Logically ANDs the results from the previous conditional statement and the next one.

Example: IF SHOT? AND VAREQ?(V30,2)

THEN

INVIS(2)

ENDIF

Object 2 will be made invisible only if the current object was shot and V30 is equal to 2

Flags: TRUE if both conditions are true

Notes:

See Also: OR, NOT

Mnemonic: ANDVAR(P1,P2) **Shorthand:** ANDV

Type: Variable Modifier

Parameters: P1 – numeric variable or constant

P2 – numeric variable

Description: Bitwise AND the contents of P1 with P2

Example: SETVAR(255,V30)

ANDVAR($3f,V30)

V30 is now equal to 63

Flags: TRUE if result is zero

Notes:

See Also: ORVAR, XORVAR, NEGVAR, NOTVAR

Mnemonic: ANIMATED? (P1 [,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1- Object number : Numeric variable or constant

P2- (Optional numeric) Area number : variable or constant

Description: Set the condition flag if object number P1, optionally in area number P2, is animated

Example:

```
IF ANIMATED?(4)
THEN
INVIS(4)
ENDIF
```

If object number 4 in the current area is animated it will be made invisible

Flags: TRUE if object is animated

Notes:

See Also: STARTANIM, STOPANIM

Mnemonic: ANIMBRUSHACTIVE?(P1) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Animbrush number : numeric variable or constant

Description: Check if animbrush number P1 is currently in use

Example:

```
IF ANIMBRUSHACTIVE?(4)
THEN
STOPANIMBRUSH(4)
ENDIF
```

If animbrush 4 is active it will be deactivated

Flags: TRUE if animbrush is active

Notes:

See Also: STARTANIMBRUSH, STOPANIMBRUSH

Mnemonic: AREAEXISTS?(P1) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Area number – numeric variable or constant

Description: Check if area number P1 exists in the current world database.

Example:

```
IF AREAEXISTS?(2)
THEN
AREANAME(S1,2)
ENDIF
```

If area number 2 exists then string S1 will be set to its name

Flags:

Notes:

Mnemonic: AREANAME(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Destn string variable

P2 – area number : numeric constant or variable

Description: Store the name of area number P2 in the string P1

Example:

```
IF AREAEXISTS?(2)
THEN
AREANAME(S1,2)
ENDIF
```

If area number 2 exists then string S1 will be set to its name

Flags:

Notes: Trying to read the name of non-existent area will produce a runtime error.

Mnemonic: BITCLEAR?(P1,P2) **Shorthand:** BC?

Type: Condition Statement

Parameters: P1 – bit number to check (0-31) : numeric variable or constant

P2 – variable to check bit in : numeric variable

Description: Check if bit number P1 is clear in variable P2

Example:

```
IF BITCLEAR?(0,V20)
THEN
ORVAR($1,V20)
ENDIF
```

If the players firing mode was disabled then enable it

Flags: TRUE if bit in given variable was clear (ie 0)

Notes: The parameter P1 is logically ANDed with 31 to ensure it is within the correct range.

See Also: BITSET?, SETBIT, CLEARBIT

Mnemonic: BITSET?(P1,P2) **Shorthand:** BS?

Type: Condition Statement

Parameters: P1 – bit number to check (0-31) : numeric variable or constant

P2 – variable to check bit in : numeric variable

Description: Check if bit number P1 is set in variable P2

Example:

```
IF BITSET?(1,V16)
THEN
PSTRING("Pressed",0,0)
ENDIF
```

If the left mouse button is pressed then the message "Pressed" is displayed in the top left hand corner of the screen

Flags: TRUE if bit in given variable was set (ie 1)

Notes: The parameter P1 is logically ANDed with 31 to ensure it is within the correct range.

See Also: BITCLEAR?, SETBIT, CLEARBIT

Mnemonic: BORDER(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – Border number – numeric variable or constant

Description: Display the border number P1

Example:

```
BORDER(1)
```

Display border number 1

Flags:

Notes: Ignores non-existent border numbers

See Also: LOADSCREEN

Mnemonic: BOX(P1,P2,P3,P4,P5) **Shorthand:**

Type: Graphics Command

Parameters: P1 – X1 posn : numeric variable or constant

P2 – Y1 Posn : numeric variable or constant

P3 – X2 posn : numeric variable or constant

P4 – Y2 Posn : numeric variable or constant

P5 – Colour : numeric variable or constant

Description: Draw a filled box whose top left hand corner is X1,Y1 and whose bottom right hand corner is X2,Y2 in colour P5

Example: BOX(0,0,319,199,5)

Will fill the entire screen with colour 5

Flags:

Notes:

See Also: FRAME, LINE, CIRCLE, DISC

Mnemonic: BRUSH (P1,P2,P3) **Shorthand:**

Type: Graphics Command

Parameters: P1 – Brush number : numeric variable or constant

P2 – X coord : numeric variable or constant

P3 – Y coord : numeric variable or constant

Description: Plot brush number P1 at the screen location P2,P3

Example: BRUSH(0,10,15)

Will plot brush 0 at location 10,15

Flags:

Notes: The coords are of the top left hand corner of the brush. Non-existent brush numbers are ignored.

See Also:

Mnemonic: CHAR(P1,P2,P3) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – source string variable or constant

P2 – Character position : numeric variable or constant

P3 – ASCII value : numeric variable

Description: Find the ASCII value of the character in the position given by P2 and save it in P3

Example: CHAR("ABCD",1,V30)

V30 will be set to $41 ("A")

Flags:

Notes:

See Also: MIDSTR, SETSTR

Mnemonic: CIRCLE(P1,P2,P3,P4) **Shorthand:**

Type: Graphics Command

Parameters: P1 – X Coord of centre: numeric variable or constant

P2 – Y Coord of centre: numeric variable or constant

P3 – radius : numeric variable or constant

P4 – colour : numeric variable or constant

Description: Draw a circle whose centre is at P1,P2 of radius P3 in colour P4

Example: CIRCLE(100,100,10,4)

Will draw a circle at coords (100,100) of radius 10 in colour 4

Flags:

Notes: In the current version, circles which are drawn outside the bounds of the screen, are not clipped properly.

See Also: FRAME, LINE, BOX, DISC

Mnemonic: CLEARARRAY(P1) **Shorthand:** CLRARY

Type: Variable Modifier

Parameters: P1 – Array variable

Description: Set every of the array pointed to by P1 to zero

Example: CLEARARRAY(A2)

All the elements in array A2 will be set to zero

Flags:

Notes:

See Also: DEFARRAY, UNDEFARRAY

Mnemonic: CLEARBIT(P1,P2) **Shorthand:** CLRB

Type: Variable Modifier

Parameters: P1 – Bit number to clear : numeric variable or constant
P2 – Destn Numeric variable

Description: Clear bit number P1 in variable P2

Example: SETVAR(255,V30)
CLEARBIT(2,V30)
V30 will be set to 251

Flags: TRUE if the result is zero

Notes: The bit number will be ANDed with 31 to ensure it is range

See Also: TOGBIT, SETBIT, BITCLEAR?, BITSET?

Mnemonic: CLEARSCREEN **Shorthand:** CLS

Type: Graphics Command

Parameters:

Description: Clear the screen

Example:

Flags:

Notes: Clears the entire screen, including any border that was loaded

See Also:

Mnemonic: CLEARSTR(P1[,P2]) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Start string variable
P2 – Optional stop string variable

Description: Set the lengths of all strings between P1 and P2 to zero. If P2 is not specified then only P1 is modified.

Example: CLEARSTR(S3,S6)
S3,S4,S5 and S6 will all be set to length zero

Flags:

Notes:

See Also: SETSTR

Mnemonic: CLEARVAR(P1[,P2]) **Shorthand:** CLRVAR

Type: Variable Modifier

Parameters: P1 – Start numeric variable

P2 – Optional stop variable

Description: Set all the variables between P1 and P2 to zero. If P2 is not specified then only P1 is cleared.

Example: CLEARVAR(V30, V35)

V30-V35 will be set to 0

Flags: TRUE if the result is zero

Notes:

See Also: SETVAR, ADDVAR, SUBVAR

Mnemonic: CLIMBABILITY(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – climb ability : numeric variable or constant

Description: Set the climb ability of the player to P1

Example: CLIMABILITY(10)

The climb ability of the player is set to 10 units

Flags:

Notes:

See Also: FALLABILITY

Mnemonic: COLLIDED? **Shorthand:** COL?

Type: Condition Statement

Parameters:

Description: Check the collision flag for the current object.

Example: IF COLLIDED?

THEN

MAKEWIRE(7)

ENDIF

If the current object has collided then make object number 7 wireframe

Flags: TRUE if object collided

Notes: Only valid on objects.

See Also: ACTIVATED?, SHOT?

Mnemonic: COS(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – angle in degrees (0-359) : numeric variable or constant

P2 – Destn variable

Description: Calculate the CoSine of P1 and store it in P2.

Example:

Flags: TRUE if result is zero

Notes: Since all maths in FCL is integer, the actual value stored in P2 is Cos(P1)*16384

See Also: SIN

Mnemonic: DATE(P1,P2,P3) **Shorthand:**

Type: IO Command

Parameters: P1 – Day variable

P2 – Month variable

P3 – Year variable

Description: Get the date from the system and put the day in P1, the month in P2 and the year in P3.

Example: DATE(V30,V31,V32)

Flags:

Notes:

See Also: TIME

Mnemonic: DEC(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Variable to decrement

Description: Decrement the variable P1

Example: SETVAR(2,V30)

DEC(V30)

V30 now equals 1

Flags: TRUE if result is zero

Notes:

See Also: INC

Mnemonic: DEFARRAY(P1,P2) **Shorthand:** DIM

Type: Variable Modifier

Parameters: P1 – Array to define

P2 – Size of required array

Description: Set the array P1 to have 10 elements

Example: DEFARRAY(A1,10)

Array A1 has 10 elements

Flags:

Notes:

See Also: UNDEFARRAY, CLEARARRAY

Mnemonic: DEFVIS?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : Numeric variable or constant

P2 – Optional area number : Numeric variable or constant

Description: Check if object number P1 in area P2 is set to visible as its default. If P2 is not specified the object is assumed to be in the current area.

Example:

```
IF DEFVIS?(4)
THEN
INVIS(4)
ENDIF
```

If object 4 in the current area is set to visible as its default, then make it invisible

Flags: TRUE if object default is visible

Notes:

See Also: VIS?

Mnemonic: DELAY(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – Frame count to delay : numeric variable or constant

Description: Pause all activity for P1 frames

Example:

Flags:

Notes:

See Also:

Mnemonic: DESTROY(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Optional area number : numeric variable or constant

Description: Mark object number P1, in area P2, as destroyed. If P2 is not specified then object is assumed to be in current area.

Example:

```
IF SOLID(4,2)?
THEN
DESTROY(4,2)
ENDIF
```

If object number 4 in area 2 is currently solid then mark it as destroyed.

Flags:

Notes:

See Also: DESTROYED?, UNDESTROY

Mnemonic: DESTROYED?(P1[,P2]) **Shorthand:** DEST?

Type: Condition Statement

Parameters: P1 – Object number : numeric constant or variable

P2 – Optional area number : numeric constant or variable

Description: Check if object number P1 in area P2 has been destroyed. If P2 is omitted then the current area is assumed.

Example:

```
IF DESTROYED?(2,1)
THEN
GOTO (1)
ENDIF
```

If object 2 in area 1 is destroyed then goto entrance 1 in this area.

Notes:

See Also: INC

Mnemonic: DEFARRAY(P1,P2) **Shorthand:** DIM

Type: Variable Modifier

Parameters: P1 – Array to define

P2 – Size of required array

Description: Set the array P1 to have 10 elements

Example: DEFARRAY(A1,10)

Array A1 has 10 elements

Flags:

Notes:

See Also: UNDEFARRAY, CLEARARRAY

Mnemonic: DEFVIS?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : Numeric variable or constant

P2 – Optional area number : Numeric variable or constant

Description: Check if object number P1 in area P2 is set to visible as its default. If P2 is not specified the object is assumed to be in the current area.

Example: IF DEFVIS?(4)

THEN

INVIS(4)

ENDIF

If object 4 in the current area is set to visible as its default, then make it invisible

Flags: TRUE if object default is visible

Notes:

See Also: VIS?

Mnemonic: DELAY(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – Frame count to delay : numeric variable or constant

Description: Pause all activity for P1 frames

Example:

Flags:

Notes:

See Also:

Mnemonic: DESTROY(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Optional area number : numeric variable or constant

Description: Mark object number P1, in area P2, as destroyed. If P2 is not specified then object is assumed to be in current area.

Example:

```
IF SOLID(4,2)?
THEN
DESTROY(4,2)
ENDIF
```

If object number 4 in area 2 is currently solid then mark it as destroyed.

Flags:

Notes:

See Also: DESTROYED?, UNDESTROY

Mnemonic: DESTROYED?(P1[,P2]) **Shorthand:** DEST?

Type: Condition Statement

Parameters: P1 – Object number : numeric constant or variable

P2 – Optional area number : numeric constant or variable

Description: Check if object number P1 in area P2 has been destroyed. If P2 is omitted then the current area is assumed.

Example:

```
IF DESTROYED?(2,1)
THEN
GOTO (1)
ENDIF
```

If object 2 in area 1 is destroyed then goto entrance 1 in this area.

Flags: TRUE if object destroyed

Notes:

See Also: DESTROY, UNDESTROY

Mnemonic: DISABLE **Shorthand:**

Type: System Modifier

Parameters:

Description: Disable all player movement

Example:

Flags:

Notes:

See Also: ENABLE

Mnemonic: DISC(P1,P2,P3,P4) **Shorthand:**

Type: Graphics Command

Parameters: P1 – X centre : numeric variable or constant

P2 – Y centre : numeric variable or constant

P3 – Radius : numeric variable or constant

P4 – Colour : numeric variable or constant

Description: Draw a filled circle at coord P1,P2 with a radius of P3 in colour P4

Example:

```
FOR (V30,10,50,10)
DISC(100,100,V30,174)
NEXT
```

Will fill an expanding disc.

Flags:

Notes:

See Also: BOX, CIRCLE, FRAME, LINE

Mnemonic: DISTANCE(P1,P2,P3,P4,P5,P6,P7) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Destination variable

P2 – X1 coord : numeric variable or constant

P3 – Y1 coord : numeric variable or constant

P4 – Z1 coord : numeric variable or constant

P5 – X2 coord : numeric variable or constant

P6 – Y2 coord : numeric variable or constant

P7 – Z2 coord : numeric variable or constant

Description: Calculate the distance between coords X1,Y1,Z1 and X2,Y2,Z2 and save it in P1

Example: DISTANCE(V30,100,100,100,75,64,800)

The distance is calculated between 100,100,100 and 75,64,800 and put into V30

Flags: TRUE if the result is zero

Notes:

See Also:

Mnemonic: DIV(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Divisor : numeric constant or variable

P2 – Dividend : numeric constant or variable

Description: Divide P2 by P1 and store the integer result in P2

Example: SETVAR(10,V30)

SETVAR(5,V31)

DIV(V31,V30)

V30 will be set to 2

Flags: TRUE if result is zero

Notes: Result is rounded down

See Also: MULT

Mnemonic: DRAWONLY **Shorthand:**

Type: System Modifier

Parameters:

Description: Forces a redraw of the Freescape view without any recalculation of the 3D view.

Example:

Flags:

Notes: Any changes in object display, such as fade value or colour changes are made, so changes in object position won't be displayed but a change in the colour of a face will.

See Also: REDRAW

Mnemonic: ELSE **Shorthand:**

Type: Condition Statement

Parameters:

Description: If the IF command associated with this ELSE was false then continue processing from here until the associated ENDIF, otherwise simply skip to the associated ENDIF.

Example:

```
VIS(2)
IF INVIS?(2) THEN
NOP
ELSE
MAKEWIRE(2)
ENDIF
```

If object 2 is made wireframe

Flags:

Notes: IF .... ENDIF blocks are nested, so it important to remember which ELSE is associated with which IF. The editor helps by automatically indenting IF...ENDIF blocks.

See Also: IF, THEN, ENDIF

Mnemonic: ENABLE **Shorthand:**

Type: System Modifier

Parameters:

Description: Enables player movement

Example:

Flags:

Notes:

See Also: DISABLE

Mnemonic: END **Shorthand:**

Type: Execution Modifier

Parameters:

Description: Unconditional stops execution of the condition.

Example:

Flags:

Notes: If use in an ANIMATOR condition, execution will recommence at the next instruction after the END, in the next frame.

See Also:

Mnemonic: ENDGAME **Shorthand:**

Type: Execution Modifier

Parameters:

Description: Forces the Freescape kernel to register the end of the game at the next frame update and perform a reset.

Example:

Flags:

Notes:

See Also:

Mnemonic: ENDIF **Shorthand:** FI

Type: Condition Statement

Parameters:

Description: Marks the end of a conditional block. The next instruction after this command will be processed, regardless of the setting of the flags.

Example:

```
VIS(4)
IF INVIS?(4)
THEN
VIS(4)
ENDIF
INVIS(4)
```

Object four will be made invisible

Flags:

Notes: As conditional blocks can be nested, it is important to keep track of which ENDIF is associated with which IF.

See Also: IF, THEN, ELSE

Mnemonic: EXECUTE(P1[,P2]) **Shorthand:** EX

Type: Execution Modifier

Parameters: P1 – object number : numeric variable or constant

P2 – optional area number : numeric variable or constant

Description: Execute the object condition associated with object P1, in area P2. If P2 is omitted then the object is assumed to be in the current area

Example:

```
IF VIS?(7,3)
THEN
EXECUTE(7,3)
ENDIF
```

If object number 7, in area 3, is visible then its conditions will be executed

Flags:

Notes:

See Also: PROC, RETURN

Mnemonic: FADE? (P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – object number : numeric variable or constant

P2 – optional area number: numeric variable or constant

Description: Check if object P1 in area P2 has faded. If P2 is omitted then object is assumed to be in the current area.

Example:

```
IF FADE?(2)
THEN
INVIS(2)
ENDIF
```

If object 2 has finished fading then make it invisible

Flags: TRUE if object has faded

Notes:

See Also: FADEIN?, FADEOUT?, FADEBOUNCE?

Mnemonic: FADEBOUNCE(P1[,P2]) **Shorthand:**
Type: Object Command
Parameters: P1 – object number : numeric variable or constant
P2 – optional area number : numeric variable or constant
Description: Set object P1 to fade in and out repeatedly.
Example: SETVAR(4,V56)
FADEBOUNCE(V56)
Object 4 will start to fade
Flags:
Notes:
See Also: FADEIN, FADEOUT, FADESTOP

Mnemonic: FADEBOUNCE?(P1[,P2]) **Shorthand:**
Type: Condition Statement
Parameters: P1 – Object number : numeric variable or constant
P2 – Optional area number : numeric variable or constant
Description: Check if object P1 in area P2 is fading in bounce mode
Example: IF FADEBOUNCE?(3)
THEN
FADESTOP(3)
ENDIF
Object 3 will stop fading
Flags: TRUE if object is in bouncing fade mode
Notes:
See Also: FADE?, FADEIN?, FADEOUT?

Mnemonic: FADEIN(P1[,P2]) **Shorthand:**
Type: Object Command
Parameters: P1 – object number : numeric variable or constant
P2 – optional area number : numeric variable or constant
Description: Set object P1 to fade in and then stop.
Example: SETVAR(4,V56)

FADEIN(V56)

Object 4 will start to fade in

Flags:

Notes:

See Also: FADEBOUNCE, FADEOUT, FADESTOP

Mnemonic: FADEIN?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : numeric variable or constant

P2 – Optional area number : numeric variable or constant

Description: Check if object P1 in area P2 is fading in.

Example:

```
IF FADEIN?(3)
THEN
DELAY(2)
ENDIF
```

If object 3 is fading in then delay the program for two frames

Flags: TRUE if object is fading in

Notes:

See Also: FADE?, FADEOUT?, FADEBOUNCE?

Mnemonic: FADEOUT(P1[,P2]) **Shorthand:**

Type: Object Command.

Parameters: P1 – object number : numeric variable or constant

P2 – optional area number : numeric variable or constant

Description: Set object P1 to fade out r.

Example:

```
SETVAR(4,V56)
FADEOUT(V56)
```

Object 4 will start to fade out.

Flags:

Notes:

See Also: FADEIN, FADEBOUNCE, FADESTOP

Mnemonic: FADEOUT?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : numeric variable or constant

P2 – Optional area number : numeric variable or constant

Description: Check if object P1 in area P2 is fading out

Example:

```
IF NOT FADEOUT?(7,3)
THEN
FADEOUT(7,3)
ENDIF
```

Object 7 in area 33 will fade out

Flags: TRUE if object is fading out

Notes:

See Also: FADE?, FADEIN?, FADEBOUNCE?

Mnemonic: FADESTOP(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – object number : numeric variable or constant

P2 – area number : numeric variable or constant

Description: Stop object P1 in area P2 from fading. If P2 is omitted then the object is assumed to be in the current area.

Example:

```
IF FADEIN?(4)
OR FADEOUT?(4)
OR FADEBOUNCE?(4)
THEN
FADESTOP(4)
ENDIF
```

If object four is undergoing any kind of fade then stop it.

Flags:

Notes:

See Also: FADEIN, FADEOUT, FADEBOUNCE

Mnemonic: FALLABILITY(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – Height – numeric variable or constant

Description: Set the fall ability of the player to P1

Example: FALLABILITY(100)

Set the player`s fall ability to 100

Flags:

Notes:

See Also: CLIMBABILITY

Mnemonic: FEXISTS?(P1) **Shorthand:**

Type: Condition Statement/IO Command

Parameters: P1 – filename string variable or constant

Description: Check if a file exists.

Example: see FOPEN for example

Flags:

Notes: If no path or drive are specified then it will assume the current directory

See Also: FOPEN, FEXISTS?, FGET, FGETS, FPUT, FPUTS

Mnemonic: FCLOSE **Shorthand:**

Type: IO Command

Parameters:

Description: Close the currently opened file

Example: see FOPEN for example

Flags:

Notes:

See Also: FOPEN, FEXISTS?, FGET, FGETS, FPUT, FPUTS

Mnemonic: FGET(P1) **Shorthand:**

Type: IO Command

Parameters: P1 – numeric variable

Description: Read a numerical value from the currently open file and store it in P1

Example: see FOPEN for example

Flags:

Notes:

See Also: FOPEN, FCLOSE,FEXISTS?, FGETS, FPUT, FPUTS

Mnemonic: FGETS(P1) **Shorthand:**

Type: IO Command

Parameters: P1 – destn string variable

Description: Read in a string from the currently opened file and store it P1

Example: see FOPEN for example

Flags:

Notes:

See Also: FOPEN, FCLOSE,FEXISTS?, FGET, FPUT, FPUTS

Mnemonic: FOPEN(P1,P2) **Shorthand:**

Type: IO Command

Parameters: P1 – filename string variable or constant

P2 – Read/write flag : numeric variable or constant

Description: Open the file pointed to by P1. P2 defines the access mode – 0=read, 1 =write.

Example:

```
IF FEXISTS?("HISCORES")
THEN
FOPEN("HISCORES",0)
FGET(V30)
FGETS(S1)
FCLOSE
ELSE
PSTRING("Hiscores not found",0,160)
ENDIF
```

If the file "HISCORES" exists then it opened and a variable and a string are read from it, otherwise a string is displayed on the screen

Flags: TRUE if successful

Notes:

See Also: FCLOSE,FEXISTS?, FGET, FGETS, FPUT, FPUTS

Mnemonic: FOR(P1,P2,P3[,P4]) **Shorthand:**

Type: Loop Command

Parameters: P1 – loop counter variable

P2 – start value : numeric variable or constant

P3 – end value : numeric variable or constant

P4 – optional increment/decrement value : numeric constant or variable

Description: Repeatedly execute the code between this FOR and its associated NEXT. When the FOR is executed P1 is set to P2. The code between the FOR and the NEXT is then executed. When the NEXT is reached 1, or P4 if specified, is added to P1. If P1 is not equal to P3 then execution jumps back to the FOR and execution continues.

Example:

```
SETVAR(0,V30)
FOR (V31,10,0,-2)
ADDVAR(1,V30)
NEXT
```

V30 will equal 5.

Flags:

Notes: Unlike LOOP ... AGAIN, FOR ... NEXT corrupts the loop variable, so, for instance, in the example above V31 will be equal to 0 when the final NEXT is executed

See Also: NEXT

Mnemonic: FPUT(P1) **Shorthand:**

Type: IO Command

Parameters: P1 – Value to write : Numeric variable or constant

Description: Write a number to the currently opened file.

Example:

Flags:

Notes:

See Also: FOPEN, FCLOSE,FEXISTS?, FGET, FGETS, FPUTS

Mnemonic: FPUTS(P1) **Shorthand:**

Type: IO Command

Parameters: P1 – output string variable or constant

Description: Write the string P1 to the currently opened file.

Example:

Flags:

Notes:

See Also: FOPEN, FCLOSE,FEXISTS?, FGET, FGETS, FPUT

Mnemonic: FRAME(P1,P2,P3,P4,P5) **Shorthand:**

Type: Graphics Command

Parameters: P1 – X1 coord : numeric variable or constant

P2 – Y1 coord : numeric variable or constant

P3 – X2 coord : numeric variable or constant

P4 – Y2 coord : numeric variable or constant

P5 – Colour to draw in : numeric variable or constant

Description: Draw a box (not filled) from P1,P2 to P3,P4 in colour P5.

Example: FRAME(0,0,319,199,2)

Will surround the entire screen in colour 2

Flags:

Notes:

See Also: BOX, CIRCLE, DISC, LINE

Mnemonic: FROMASCII(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – source string variable or constant

P2 – destn numeric variable

Description: Store the numeric value of string P1 in variable P2

Example: FROMASCII("100",V30)

V30 will now be set to 100

Flags:

Notes:

See Also: TOASCII

Mnemonic: GETFADE(P1,P2,P3) **Shorthand:**

Type: Object Command

Parameters: P1 – destn variable

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Get the current fade value of object number P2 in area P1 and put it into P1. If P2 is omitted then the current area is assumed

Example: GETFADE(V30,2)

Sets V30 to the fade value of object 2

Flags:

Notes:

See Also: SETFADE

Mnemonic: GETFADER(P1,P2,P3) **Shorthand:**

Type: Object Command

Parameters: P1 – Destn numeric variable

P2 – User fade number : numeric variable or constant

P3 – Line of fade pattern : numeric variable or constant

Description: Get the current bitmask for line P3 in user fade number P2 and put it in variable P1.

Example: GETFADER(V30,12,2)

V30 will be set to the value of line 2 in user fade 12.

Flags:

Notes: The user fades start at 10 and end at 15.

See Also: SETFADER

Mnemonic: GETOBJCOL(P1,P2,P3[,P4]) **Shorthand:**

Type: Object Command

Parameters: P1 – Destn variable

P2 – Facet number : numeric variable or constant

P3 – Object number : numeric variable or constant

P4 – Optional area number : numeric variable or constant

Description: Set P1 to the colour of facet number P2 in object P3, which is area P4. If O4 is omitted then the current area is assumed

Example: IF VIS?(4,2)

THEN GETOBJCOL(V30,1,4,2)

V30 is set to the colour of facet 1 of object 4 in area 2

Flags:

Notes:

See Also: SETOBJCOL

Mnemonic: GETPIXEL(P1,P2,P3) **Shorthand:**

Type: Object Command

Parameters: P1 – X coord : numeric variable or constant

P2 – Y coord : numeric variable or constant

P3 – Destn variable

Description: Get the colour of the pixel at location P1,P2 and put it in P3.

Example: GETPIXEL(10,10,V43)

V43 will be set to the colour of the pixel at location 10,10 on the screen

Flags:

Notes:

See Also: SETPIXEL

Mnemonic: GETXPOS(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – Destn variable

P2 – Object number : numeric variable or constant

P3 – Optional area number : numeric variable or constant

Description: Get X position of object number P2 in area P3 and store in P1. If P3 is omitted then the current area is assumed.

Example: GETXPOS(V30,2)

V30 is set to the world X coord of object 2

Flags:

Notes:

See Also: SETXPOS

Mnemonic: GETXSIZE (P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – Destn variable

P2 – Object number : numeric variable or constant

P3 – Optional area number : numeric variable or constant

Description: Get X size of object number P2 in area P3 and store in P1. If P3 is omitted then the current area is assumed.

Example: GETXSIZE(V34,4,1)

V34 is set to the size in the X axis of object 4 in area 1

Flags:

Notes:

See Also: SETXSIZE

Mnemonic: GETYPOS(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – Destn variable

P2 – Object number : numeric variable or constant

P3 – Optional area number : numeric variable or constant

Description: Get Y position of object number P2 in area P3 and store in P1. If P3 is omitted then the current area is assumed.

Example: GETYPOS(V30,2)

V30 is set to the world Y coord of object 2

Flags:

Notes:

See Also: SETYPOS

Mnemonic: GETYSIZE (P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – Destn variable

P2 – Object number : numeric variable or constant

P3 – Optional area number : numeric variable or constant

Description: Get Y size of object number P2 in area P3 and store in P1. If P3 is omitted then the current area is assumed.

Example: GETYSIZE(V34,4,1)

V34 is set to the size in the Y axis of object 4 in area 1

Flags:

Notes:

See Also: SETYSIZE

Mnemonic: GETZPOS(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – Destn variable

P2 – Object number : numeric variable or constant

P3 – Optional area number : numeric variable or constant

Description: Get Z position of object number P2 in area P3 and store in P1. If P3 is omitted then the current area is assumed.

Example: GETZPOS(V30,2)

V30 is set to the world Z coord of object 2

Flags:

Notes:

See Also: SETZPOS

Mnemonic: GETZSIZE (P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – Destn variable

P2 – Object number : numeric variable or constant

P3 – Optional area number : numeric variable or constant

Description: Get Z size of object number P2 in area P3 and store in P1. If P3 is omitted then the current area is assumed.

Example: GETZSIZE(V34,4,1)

V34 is set to the size in the Z axis of object 4 in area 1

Flags:

Notes:

See Also: SETYSIZE

Mnemonic: GOBACK **Shorthand:**

Type: System Modifier

Parameters:

Description: Move the player back to the last visited area at the last position in that area.

Example:

Flags:

Notes:

See Also: GOTO

Mnemonic: GOTO(P1[,P2]) **Shorthand:**

Type: System Modifier

Parameters: P1 – entrance number : numeric variable or constant

P2 – optional area number : numeric variable or constant

Description: Move the player to entrance P1 in area P2: If P2 is omitted then the current area is assumed

Example:

```
IF SHOT?
THEN
GOTO(2)
ENDIF
```

If the current object has been shot then go to entrance 2 in the current area

Flags:

Notes:

See Also: GOBACK

Mnemonic: GOTOCAMERA(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – camera number : numeric variable or constant

Description: Move the player to the current position of Camera P1

Example:

```
GOTOCAMERA(2)
```

The players position will be moved to camera 2

Flags:

Notes:

See Also:

Mnemonic: IF **Shorthand:**

Type: Condition Statement

Parameters:

Description: Marks the beginning of a conditional block and clears the condition flags ready for a conditional statement.

Example:

Flags:

Notes:

See Also: THEN, ELSE, ENDIF

Mnemonic: INC(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – numeric variable

Description: Adds one to P1

Example: SETVAR(0,V30)

INC(V30)

V30 will equal 1

Flags: TRUE if result is zero

Notes:

See Also: DEC

Mnemonic: INCLUDE(P1) **Shorthand:**

Type: Animator Command

Parameters: P1 – object number : numeric variable or constant

Description: Add object P1 to list of object affected by this animator.

Example: INCLUDE(2)

Object number 2 will now be included in this animation

Flags:

Notes: Only valid in ANIMATOR Conditions

See Also: REMOVED, ANIMATED?

Mnemonic: INDEXSTR(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – source string number : numeric variable or constant

P2 – Destn string variable

Description: Store string number P1 in string P2

Example: SETSTR("Hello",S1)

SETSTR("World",S2)

SETVAR(2,V30)

INDESTR(V30,S1)

S1 will be set to "World"

Flags:

Notes:

See Also:

Mnemonic: INSCOL(P1,P2,P3) **Shorthand:**

Type: Graphics Command

Parameters: P1 – instrument number : numeric variable or constant

P2 – Foreground colour : numeric variable or constant

P3 – Background colour : numeric variable or constant

Description: Set the foreground colour to P1 and the background colour to P2 for instrument P3

Example: INSCOL(1,4,7)

The foreground colour of instrument 1 is set to colour 4 and the background colour is set to 7

Flags:

Notes:

See Also:

Mnemonic: INSDISABLE(P1) **Shorthand:**

Type: Graphics Command

Parameters: P1 – Instrument number : numeric variable or constant

Description: Disable instrument P1

Example: INSDISABLE(4)

Instrument number 4 will no longer be updated

Flags:

Notes:

See Also: INSENABLE

Mnemonic: INSENABLE(P1) **Shorthand:**

Type: Graphics Command

Parameters: P1 – Instrument number : numeric variable or constant

Description: Disable instrument number P1

Example: INSENABLE(2)

Instrument 2 will be updated

Flags:

Notes:

See Also: INSDISABLE

Mnemonic: INTAN?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Check if object P1 in area P2 is intangible

Flags:

Notes:

See Also: TAN?, MAKETAN, MAKEINTAN

Mnemonic: INVIS(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Optional area number : numeric variable or constant

Description: Set the object attributes for object P1 in area P2 to make it invisible. If P2 is omitted then the current area is assumed.

Example:

```
IF SHOT?
THEN
INVIS(1,4)
ENDIF
```

If the current object has been shot then make object 1 in area 4 invisible

Flags:

Notes:

See Also: VIS, VIS?, INVIS?

Mnemonic: INVIS?(P1[,P2]) **Shorthand:**

Type:

Parameters: P1 – Object number : numeric variable or constant

P2 – Optional area number : numeric variable or constant

Description: Set the object attributes for object P1 in area P2 to make it invisible. If P2 is omitted then the current area is assumed.

Example:

```
IF INVIS?(7)
THEN
MOVE(20,0,0)
ENDIF
```

If object 7 is invisible then move the animation list set in this animator

Flags: TRUE if object is invisible

Notes:

See Also: INVIS, VIS, VIS?

Mnemonic: JUMP(:P1) **Shorthand:**

Type: Execution Modifier

Parameters: P1 – label number

Description: Transfer execution to the label :P1

Example: IF SHOT?

THEN

JUMP (:1)

ENDIF

INVIS(2)

:1 INVIS(3)

Will only make object 2 invisible if the current object was shot. Object 3 will always be made invisible

Flags:

Notes:

See Also:

Mnemonic: LEFTSTR(P1,P2,P3) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Source string variable or constant

P2 – Count : numeric constant or variable

P3 – Destination string variable

Description: Set the string P3 to be equal to the first P2 characters in P1

Example: SETSTR("Hello World",S1)

LEFTSTR(S1,5,S2)

S2 will be set to "Hello"

Flags:

Notes:

See Also: RIGHTSTR, MIDSTR

Mnemonic: LENSTR(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – source string variable

P2 – destn numeric variable

Description: Calculate the length of the string P1 and save it in variable P2

Example: SETSTR("Hello World",S1)

LENSTR(S1,V30)

V30 will be set to 11

Flags:

Notes:

See Also:

Mnemonic: LIMIT(P1,P2,P3,P4) **Shorthand:**

Type: Graphics Command

Parameters: P1 – Destn numeric variable

P2 – Source numeric variable or constant

P3 – Min : numeric variable or constant

P4 – Max : numeric variable or constant

Description: Limit the value of P2 to between P3 and P4 and store the result in P1.

Example: SETVAR(10,V31)

LIMIT(V30,V31,0,8)

V30 will be set to 8.

Flags: TRUE if result is zero

Notes:

See Also: MAX, MIN

Mnemonic: LINE(P1,P2,P3,P4,P5) **Shorthand:**

Type: Graphics Command

Parameters: P1 – X1 coord : numeric variable or constant

P2 – Y1 coord : numeric variable or constant

P3 – X2 coord : numeric variable or constant

P4 – Y2 coord : numeric variable or constant

P5 – Line colour : numeric variable or constant

Description: Draw a line on the screen between (P1,P2) and (P3,P4) in colour P5

Example: LINE(0,0,319,0,23)

Will draw a line in colour 23 across the very top line of the screen

Flags:

Notes:

See Also: BOX, CIRCLE, DISC, FRAME

Mnemonic: LOADSCREEN(P1,P2) **Shorthand:**

Type: IO Command/Graphics Command

Parameters: P1 – filename to load : string variable or constant

P2 – Display mode : numeric variable or constant

Description: Load and display the screen whose file specification is P1. P2 is the display mode : 0 Just display picture, do not put 3D window up

1 Display picture with 3D window

Example: LOADSCREEN("PANEL.IFF",0)

Will load and display the file called PANEL.IFF

Flags:

Notes:

See Also: BORDER

Mnemonic: LOADWORLD(P1) **Shorthand:**

Type: IO Command/System Modifier

Parameters: P1 – filename of world : string variable or constant

Description: Load the world data file called P1.

Example: LOADWORLD("my_world.3wd")

Will load the worlds datefile called MY_WORLD.3WD

Flags:

Notes: Will overwrite any existing data in memory.

See Also:

Mnemonic: LOCK(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – object number : numeric variable or constant

P2 – optional area number : numeric variable or constant

Description: Lock object P1 in area P2 against being used in animation commands. If P2 is not specified then the current area is assumed

Example: LOCK(4)

Object 4 will no longer be animated

Flags:

Notes:

See Also: LOCKED?, UNLOCK, UNLOCKED?

Mnemonic: LOCKED?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : numeric variable or constant

P2 – Optional area number : numeric variable or constant

Description: Check if object P1 in area P2 is locked against animation. If P2 is not specified then the current area is assumed.

Example:

```
IF LOCKED?(3)
THEN
UNLOCK(3)
ENDIF
```

Flags: TRUE if object is locked against animation

Notes:

See Also: LOCKED, UNLOCK, UNLOCKED?

Mnemonic: LOCKONTO(P1,P2,P3[,P4]) **Shorthand:**

Type: System Modifier

Parameters: P1 – view direction : numeric variable or constant

P2 – view distance : numeric variable or constant

P3 – Object number to lock to : numeric variable or constant

P4 – optional area number : numeric variable or constant

Description: Move the player`s viewpoint such that the centre of object P3, in area P4, is in the centre of the screen and the players eye position is facing the direction specified by P1 at the distance specified by P2. If P4 is not specified then current area is assumed.

The view direction, P1, takes the following values :

0 – North

1 – West

2 – East

3 – South

4 – Above

5 – Below

Example: LOCKONTO(1,500,2)

Moves the player viewpoint to face object 2 from the West at a distance of 500 units

Flags:

Notes:

See Also:

Mnemonic: LOOP(P1) **Shorthand:**

Type: Loop Command

Parameters: P1 – Repeat count : numeric constant or variable

Description: Execute the code between this LOOP and its associated AGAIN command P1 times.

Example: SETVAR(0,V30)

LOOP(10)

ADDVAR(3,V30)

AGAIN

V30 will have the value of 30

Flags:

Notes: Unlike FOR...NEXT, once the LOOP command has been executed then the parameter P1 is no longer needed, so if it is a variable, it may be re-used inside the loop, and if it is not altered then it will have the same value after the LOOP...AGAIN has finished as it did before it started. For instance :

SETVAR(5,V30)

LOOP(V30)

ADDVAR(1,V30)

AGAIN

is a valid piece of code and will result in V30 having the value 10 when the loop terminates.

See Also: AGAIN

Mnemonic: MAKEINTAN(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Set the object attributes for object P1 in area P2 so that it is intangible. If P2 is not specified then the current area is assumed.

Example: MAKEINTAN(2)

Object 2 will be made intangible.

Flags:

Notes:

See Also: MAKETAN, TAN?, INTAN?

Mnemonic: MAKESOLID(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Set the object attributes for object P1 in area P2 so that it is solid (ie not wireframe). If P2 is not specified then the current area is assumed.

Example: MAKESOLID(2)

Object 2 will be made solid.

Flags:

Notes:

See Also: MAKEWIRE, WIRE?

Mnemonic: MAKETAN(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Set the object attributes for object P1 in area P2 so that it is tangible. If P2 is not specified then the current area is assumed.

Example: MAKETAN(2)

Object 2 will be made tangible.

Flags:

Notes:

See Also: MAKEINTAN, TAN?, INTAN?

Mnemonic: MAKEWIRE(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Set the object attributes for object P1 in area P2 so that it is wireframe. If P2 is not specified then the current area is assumed.

Example: MAKEWIRE(2)

Object 2 will be made wireframe.

Flags:

Notes:

See Also: MAKESOILD, WIRE?

Mnemonic: MIDSTR(P1,P2,P3,P4) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Source string variable or constant

P2 – Start posn : numeric variable or constant

P3 – End posn : numeric variable or constant

P4 – Destn string variable

Description: P4 is set to a sub-string of P1, starting at character P2 and ending at character P3.

Example: MIDSTR("Hello World",5,7,S1)

S1 is set to "o W"

Flags:

Notes:

See Also:

Mnemonic: MAX(P1,P2,P3) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – destn numeric variable

P2 – source numeric variable or constant

P3 – Max value : numeric variable or constant

Description: P1 is set to the value of P2, limited so that the maximum value it can have is P3.

Example: SETVAR(24,V30)

MAX(V31,V30,12)

V30 will be set to twelve

Flags: TRUE if result is zero

Notes:

See Also: LIMIT, MIN

Mnemonic: MIN(P1,P2,P3) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – destn numeric variable

P2 – source numeric variable or constant

P3 – Min value : numeric variable or constant

Description: P1 is set to the value of P2, limited such that the minimum value it can have is P3.

Example: SETVAR(-24,V30)

MIN(V31,V30,12)

V30 will be set to twelve

Flags: TRUE if result is zero

Notes:

See Also: LIMIT, MAX

Mnemonic: MODE(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – movement mode : numeric variable or constant

Description: Change the player`s movement mode to P1. P1 can be

1 – walk, 2 – fly1, 3- fly2.

Example: MODE(2)

The player`s movement mode is set to fly1.

Flags:

Notes:

See Also:

Mnemonic: MOVE(P1,P2,P3) **Shorthand:**

Type: Animator Command

Parameters: P1 – delta X change : numeric variable or constant

P2 – delta Y change : numeric variable or constant

P3 – delta Z change : numeric variable or constant

Description: Move the animation list, for this ANIMATOR, by the relative amount in each axis.

Example: MOVE(5,7,-120)

The current animation list will be move 5 units along the X axis, 7 units along the Y axis and -120 units along the Z axis.

Flags:

Notes: Only valid in ANIMATORS

See Also: MOVETO

Mnemonic: MOVEABLE?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – object number : numeric variable or constant

P2 – area number : numeric variable or constant

Description: Check the object attribute for object P1, in area P2, to see if it is moveable. If P2 is not specified then the current area is assumed.

Example: IF MOVEABLE?(3)

THEN

SETVAR(3,V200)

STARTANIM(1)

ENDIF

If object 3 is moveable then set V200 to 3 and start ANIMATOR 3

Flags: TRUE if object moveable

Notes:

See Also:

Mnemonic: MOVETO(P1,P2,P3) **Shorthand:**

Type: Animator Command

Parameters: P1 – New X Position : numeric variable or constant
P2 – New Y Position: numeric variable or constant
P3 – New Z Position: numeric variable or constant

Description: Move the animation list to the absolute coords P1,P2,P3.

Example: MOVE(0,0,0)
The current animation list will be moved to 0,0,0

Flags:

Notes: Only valid in ANIMATORS

See Also: MOVE

Mnemonic: MULT(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – numeric variable or constant
P2 – Numeric variable

Description: Calculate P1*P2 and put the result in P2

Example: SETVAR(20,P1)
MULT(10,P1)
P1 will be set to 200 .

Flags: TRUE if result is zero

Notes:

See Also: DIV

Mnemonic: NEGATIVE?(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – numeric variable or constant

Description: Check to see if P1 is less than zero

Example:
```
SETVAR(-1,V30)
IF NEGATIVE?(V30)
THEN
PSTRING("Hello",0,0)
ENDIF
```
The string "Hello" will be printed in the top left hand corner of the screen.

Flags: TRUE if P1 is negative.
Notes:
See Also:

Mnemonic: NEGVAR(P1) **Shorthand:**
Type: Variable Modifier
Parameters: P1 – numeric variable
Description: Negate the value in P1 and put it back in P1
Example: SETVAR(177,V34)
NEGVAR(V34)
V34 will contain -177.
Flags: TRUE if result is zero.
Notes:
See Also: NOTVAR, XORVAR, ANDVAR, ORVAR

Mnemonic: NEXT **Shorthand:**
Type: Loop Command
Parameters:
Description: Marks the end of a FOR...NEXT.
Example:
Flags:
Notes: See the definition of FOR for examples and description
See Also: FOR

Mnemonic: NOP **Shorthand:**
Type: Execution Modifier
Parameters:
Description: No operation. Ignored by FCL.
Example: IF VIS?(3)
THEN
NOP
ELSE

VIS(3)

ENDIF

If object 3 is visible then do nothing else make it visible.

Flags:

Notes:

See Also:

Mnemonic: NOT **Shorthand:** !

Type: Condition Statement

Parameters:

Description: Logically inverts the result of the following condition.

Example: IF NOT ADDVAR(4,V55)

THEN

MOVE(400,0,0)

ENDIF

4 is added to V55. If the result is not 0 then move the current animation list 400 units along the X axis.

Flags: TRUE if resulting condition is FALSE.

Notes:

See Also: OR, AND

Mnemonic: NOTVAR(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – numeric variable

Description: Bitwise complement variable P1. This is not the same as negation.

Example: SETVAR(0,V77)

NOTVAR(V77)

V77 will contain -1

Flags: TRUE if result is zero.

Notes:

See Also: ORVAR, XORVAR, ANDVAR, NEGVAR

Mnemonic: OBJEXISTS?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1- object number : numeric variable or constant

P2 – area number : numeric variable or constant

Description: Check if object number P1 in area P2 actually exists. If P2 is not specified then the current area is assumed.

Example:

```
IF OBJEXISTS?(74)
THEN
INVIS(74)
ENDIF
```

If object 74 exists then make it invisible.

Flags: TRUE if object exists

Notes:

See Also:

Mnemonic: OBJNAME(P1,P2[,P3]) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Destn string variable

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Store the name of object P2 in area P3 in the string P1. If P3 is not specified then the current area is assumed.

Example:

```
OBJNAME(S1,5)
```

The name of object number 5 is stored in the string variable S1.

Flags:

Notes:

See Also:

Mnemonic: OR **Shorthand:** ||

Type: Condition Statement

Parameters:

Description: Logically ORs the results from the previous conditional statement with next conditional statement.

Example: IF INVIS?(4) OR INVIS?(5)
THEN
INVIS(6)
ENDIF
If object 4 or object 5 are invisible then make object 6 invisible

Flags: TRUE if either result is TRUE

Notes:

See Also: IF, AND, NOT

Mnemonic: ORVAR(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – numeric constant or variable
P2 – numeric variable

Description: Bitwise OR the contents of P1 with P2 and put the result in P2.

Example: SETVAR($55,V30)
ORVAR($AA,V30)
V30 will contain $FF

Flags: TRUE if result is zero

Notes:

See Also: ANDVAR, XORVAR, NEGVAR, NOTVAR

Mnemonic: PAUSE(P1) **Shorthand:**

Type: Execution Modifier

Parameters: P1 – Key value : numeric variable or constant

Description: Waits for the key whose ASCII value is P1 to be pressed. If P1 is zero then the program waits for any key to be pressed.

Example: PAUSE($41)
Wait for "A" to be pressed

Flags:

Notes:

See Also:

Mnemonic: PCHAR(P1,P2,P3) **Shorthand:**

Type: Graphics Command

Parameters: P1 – ASCII character value : numeric variable or constant

P2 – X coord : numeric variable or constant

P3 – Y coord : numeric variable or constant

Description: Display the character whose ASCII value is P1 and display it at the coord P2,P3 on the screen.

Example: PCHAR($31,34,65)

Will display "1" at location 34,65

Flags:

Notes:

See Also: PNUMBER, PSTRING, TEXTFONT, TEXTCOL

Mnemonic: PNUMBER(P1, P2, P3,P4)**Shorthand:**

Type: Graphics Command

Parameters: P1 – numeric variable

P2 – print length : numeric constant or variable

P3 – X coord : numeric constant or variable

P4 – Y coord : numeric constant or variable

Description: Print the number P1 in a field of length P2 characters, at screen position P3,P4

Example: SETVAR(34567,V30)

PNUMBER(V30,5,0,0)

"34567" is printed at location 0,0

Flags:

Notes: If bit 15 is set in the length parameter (add $8000 to value) then leading zeros will be displayed in the field to pad out any unused positions. If bit 14 is set in the length parameter (add $4000 to value) then the sign of the number will not be displayed. If the length parameter is not long enough to hold the whole number then the least significant digits are printed

eg

1234567 in a field of length 4 will be printed as 4567.

See Also: PCHAR, PSTRING, TEXTFONT, TEXTCOL

Mnemonic: PRINT(P1,P2) **Shorthand:**

Type: Graphics Command

Parameters: P1- string variable or constant

P2 – instrument number

Description: Print the string P1 into instrument number P2

Example: SETSTR("Hello",S1)

PRINT(S1,2)

"Hello" will be printed to instrument 2

Flags:

Notes:

See Also:

Mnemonic: PROC(P1) **Shorthand:**

Type: Execution Modifier

Parameters: P1 – procedure number : numeric variable or constant

Description: Save the current execution address and jump to the procedure number P1. When the procedure is finished, execution will return to the first instruction after the PROC instruction.

Example: PROC(10)

Procedure number 10 will be called

Flags:

Notes:

See Also: EXECUTE, RETURN

Mnemonic: PSTRING(P1,P2,P3) **Shorthand:**

Type: Graphics Command

Parameters: P1 – string variable or constant

P2 – X coord : numeric variable or constant

P3 – Y coord : numeric variable or constant

Description: Print the string P1 on the screen at screen location P2,P3.

Example: PSTRING("Hello",0,0)

"Hello" will be printed at location 0,0 on the screen.

Flags:

Notes:

See Also: PCHAR, PNUMBER, TEXTFONT, TEXTCOL

Mnemonic: RANDOM(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Maximum value : numeric variable or constant

P2 – Destn numeric variable

Description: Place a random number between 0 and P1 in P2.

Example: RANDOM(10,V34)

Flags: TRUE if random number is zero

Notes:

See Also:

Mnemonic: REDRAW **Shorthand:**

Type: System Modifier

Parameters:

Description: Force a redraw of the Freescape world at the next frame.

Example:

Flags:

Notes:

See Also: DRAWONLY

Mnemonic: REMOVE(P1) **Shorthand:**

Type: Animator Command

Parameters: P1 – Object number : numeric variable or constant

Description: Remove object P1 from the animation list for this animator.

Example: REMOVE(10)

Object 10 will be removed from the animation list

Flags:

Notes: Only valid in ANIMATORS.

See Also: INCLUDE

Mnemonic: RESETAREA(P1) **Shorthand:**

Type: System Modifier

Parameters: P1 – Area number : numeric variable or constant

Description: Force a reset of all the data associated with area P1.

Example: RESETAREA(2)

Area 2 will be reset

Flags:

Notes:

See Also: RESETOBJ

Mnemonic: RESETOBJ(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Force object P1, in area P2, to be reset to its default settings for position, attributes and colours. Also resets the object's detection flags. If P2 is not specified then the object is assumed to be in the current area.

Example: RESETOBJ(3,1)

Object number 3 in area 1 will be reset

Flags:

Notes:

See Also: RESETAREA, RESETOBJCOL, RESETOBJPOS

Mnemonic: RESETOBJCOL(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Force object P1, in area P2, to be reset to its default settings for colours. If P2 is not specified then the object is assumed to be in the current area.

Example: RESETOBJCOL(3,1)

Object number 3 in area 1 will be reset to its default colours

Flags:

Notes:

See Also: RESETOBJ, RESETOBJPOS

Mnemonic: RESETOBJPOS(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Force object P1, in area P2, to be reset to its default settings for position. If P2 is not specified then the object is assumed to be in the current area.

Example: RESETOBJPOS(3,1)

Object number 3 in area 1 will be reset to its default position.

Flags:

Notes:

See Also: RESETOBJ, RESETOBJCOL

Mnemonic: RESTART **Shorthand:**

Type: Execution Modifier

Parameters:

Description: Return the execution position to the marked START position in an ANIMATOR. If no START position was marked then execution restarts at the beginning of the ANIMATOR.

Example:

Flags:

Notes: Only valid in ANIMATORS.

See Also: START

Mnemonic: RETURN **Shorthand:**

Type: Execution Modifier

Parameters:

Description: Return from a procedure or object. If returning to a procedure then execution continues after the PROC statement.

Example:

Flags:

Notes:

See Also: PROC, EXECUTE

Mnemonic: RIGHTSTR(P1,P2,P3) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – Source string or variable

P2 – String length : numeric constant or variable

P3 – Destination string variable

Description: Store the last P2 characters from the source string P1 in the destination string P3.

Example: RIGHTSTR("Hello World",5,S1)

S1 will be set to "World"

Flags:

Notes:

See Also:

Mnemonic: ROOT(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – source number : numeric constant or variable

P2 – destn numeric variable

Description: Calculate the square root if P1 and save it in P2.

Example: ROOT(4,V30)

V30 will be set to 2

Flags: TRUE if result is zero

Notes:

See Also:

Mnemonic: SETBIT(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – bit position : numeric variable or constant

P2 – destn numeric variable

Description: Set bit number P1 in variable P2.

Example: SETVAR(0,V30)

SETBIT(4,V30)

V30 is set to 16.

Flags: TRUE if the result is zero

Notes:

See Also: TOGBIT, CLEARBIT, BITSET?, BITCLEAR?

Mnemonic: SETFADE(P1,P2[,P3]) **Shorthand:**

Type: Graphics Command

Parameters: P1 – Fade value : numeric variable or constant

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Set the fade value of object number P2, in area P3, to P1. If P3 is not specified then the current area is assumed.

Example: SETFADE(2,4,6)

Object number 4 in area 6 has its fade value set to 2.

Flags:

Notes:

See Also: GETFADE

Mnemonic: SETFADER(P1,P2,P3) **Shorthand:**

Type: Graphics Command

Parameters: P1 – User fade number : numeric variable or constant

P2 – Line of mask to change : numeric variable or constant

P3 – New mask value : numeric variable or constant

Description: Set line P2 of the user fade number P1 to the bit image specified by P3.

Example: SETFADER(10,1,$AA)

Line 1 of user fade 10 will be set to a cross-hatch.

Flags:

Notes: The user fades start at 10 and end at 15, you cannot edit the system fade values from 0 to 9.

See Also: GETFADER

Mnemonic: SETGROUND(P1) **Shorthand:**

Type: Graphics Command

Parameters: P1 – New ground colour : numeric variable or constant

Description: Set the ground colour to be colour number P1

Example: SETGROUND(23)

The ground will now be colour 23

Flags:

Notes:

See Also: SETSKY

Mnemonic: SETOBJCOL(P1,P2,P3[,P4]) **Shorthand:**

Type: Object Command

Parameters: P1 – Colour number : numeric variable or constant

P2 – Facet number : numeric variable or constant

P3 – Object number : numeric variable or constant

P4 – Area number : numeric variable or constant

Description: Set facet number P2 of object number P3, in area P4, to colour number P1. If the area number is omitted then the current area is assumed.

Example: SETOBJCOL(2,1,7)

Facet 1 of object 7 will be set to colour 2

Flags:

Notes:

See Also: GETOBJCOL

Mnemonic: SETPIXEL(P1,P2,P3) **Shorthand:**

Type: Graphics Command

Parameters: P1 – X coord : numeric variable or constant

P2 – Y coord : numeric variable or constant

P3 – Colour number : numeric variable or constant

Description: Set the pixel at screen location P1,P2 to colour P3.

Example: SETPIXEL(34,123,234)

The pixel at screen coords (34,123) will be set to colour 234

Flags:
Notes:
See Also: GETPIXEL

Mnemonic: SETSKY(P1) **Shorthand:**
Type: Graphics Command
Parameters: P1 – New sky colour : numeric variable or constant
Description: Set the sky colour to be colour number P1
Example: SETSKY(177)
The sky will now be colour 177
Flags:
Notes:
See Also: SEYGROUND

Mnemonic: SETSTR(P1,P2) **Shorthand:**
Type: Variable Modifier
Parameters: P1 – source string constant or variable
P2 – destn string variable
Description: Set string P2 to be equal to string P1.
Example: SETSTR("Hello World",S4)
S4 will now be "Hello World"
Flags:
Notes:
See Also: ADDSTR, CLEARSTR

Mnemonic: SETVAR(P1,P2) **Shorthand:**
Type: Variable Modifier
Parameters: P1 – source number : numeric variable or constant
P2 – destn numeric variable
Description: Set P2 to the value of P1.
Example: SETVAR(-17765,V237)
V237 will now have the value V237

Flags: TRUE if result is zero

Notes:

See Also: ADDVAR, SUBVAR, CLEARVAR

Mnemonic: SETXPOS(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – New X position : numeric variable or constant

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Set the X position of object P2, in area P3, to P1. If P3 is omitted then the object is assumed to be in the current area.

Example: SETXPOS(100,2)

Object 2 in the current area will have its X position set to 100

Flags: TRUE if position is at zero

Notes:

See Also: GETXPOS

Mnemonic: SETXSIZE(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – New X size: numeric variable or constant

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Set the the size of object P2, in area P3, to P1 in the X axis. If P3 is omitted then the object is assumed to be in the current area.

Example: SETXSIZE(100,2)

Object 2 in the current area will have its X size set to 100

Flags: TRUE if size is zero

Notes:

See Also: GETXSIZE

Mnemonic: SETYPOS(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – New Y position : numeric variable or constant

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Set the Y position of object P2, in area P3, to P1. If P3 is omitted then the object is assumed to be in the current area.

Example: SETYPOS(100,2)

Object 2 in the current area will have its Y position set to 100

Flags: TRUE if position is at zero

Notes:

See Also: GETYPOS

Mnemonic: SETYSIZE(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – New Y size: numeric variable or constant

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Set the the size of object P2, in area P3, to P1 in the Y axis. If P3 is omitted then the object is assumed to be in the current area.

Example: SETYSIZE(100,2)

Object 2 in the current area will have its X size set to 100

Flags: TRUE if size is zero

Notes:

See Also: GETYSIZE

Mnemonic: SETZPOS(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – New Z position : numeric variable or constant

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Set the Z position of object P2, in area P3, to P1. If P3 is omitted then the object is assumed to be in the current

area.

Example: SETZPOS(100,2)

Object 2 in the current area will have its Z position set to 100

Flags: TRUE if position is at zero

Notes:

See Also: GETZPOS

Mnemonic: SETZSIZE(P1,P2[,P3]) **Shorthand:**

Type: Object Command

Parameters: P1 – New Z size: numeric variable or constant

P2 – Object number : numeric variable or constant

P3 – Area number : numeric variable or constant

Description: Set the the size of object P2, in area P3, to P1 in the Z axis. If P3 is omitted then the object is assumed to be in the current area.

Example: SETZSIZE(100,2)

Object 2 in the current area will have its Z size set to 100

Flags: TRUE if size is zero

Notes:

See Also: GETZSIZE

Mnemonic: SHIFTLEFT(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – numeric variable

Description: Shift P1 left one bit and, shifting a 0 bit into the least significant bit. This is functionally equivalent to multiplying the P1 by 2.

Example: SETVAR(1,V77)

SHIFTLEFT(V77)

V77 will now be 2

Flags: TRUE if most significant bit set

Notes: Warning: shifting a negative number left can have unpredictable results. It is much better to convert the number to a positive one, shift it, and then convert it back

to a negative number

eg

```
IF VARLT?(0,V30)
THEN
NEGVAR(V30)
SHIFTLEFT(V30)
NEGVAR(V30)
ELSE
SHIFTLEFT(V30)
ENDIF
```

See Also: SHIFTRIGHT

Mnemonic: SHIFTRIGHT(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 : numeric variable

Description: Shift the variable P1 right by one bit, sign extending an necessary (ie if the number is negative then a 1 is shifted in, else a 0 is shifted in).

Example:

```
SETVAR(23,V30)
SHIFTRIGHT(V30)
```

V30 will now have the value 11

Flags: TRUE if least significant bit set

Notes: Functionally equivalent to dividing by 2

See Also: SHIFTLEFT

Mnemonic: SHOT? **Shorthand:**

Type: Condition Statement

Parameters:

Description: Test to see if the current object was shot (ie activated with the LEFT mouse button).

Example:

```
IF SHOT?
THEN
INVIS(2)
ENDIF
```

If the current object was shot then set object 2 invisible.

Flags: TRUE if object was shot.

Notes: Only valid in object conditions

See Also: ACTIVATED?, COLLIDED?

Mnemonic: SIN(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – angle in degrees (0-359) : numeric variable or constant

P2 – Destn variable

Description: Calculate the Sine of P1 and store it in P2.

Example:

Flags: TRUE if result is zero

Notes: Since all maths in FCL is integer, the actual value stored in P2 is Sin(P1)*16384

See Also: COS

Mnemonic: SOLID?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – object number : numeric variable or constant

P2 – area number : numeric variable or constant

Description: Check the object attributes for object P1, in area P2, to see if the object is currently solid. If P2 is omitted then the object is assumed to be in the current area.

Example:

```
IF SOLID?(4,5)
THEN
MAKEWIRE(4,5)
ENDIF
```

If object 4 in area 5 is solid then make it wireframe.

Flags: TRUE if object is solid.

Notes:

See Also: WIRE?, MAKESOLID, MAKEWIRE

Mnemonic: SOUND(P1) **Shorthand:**

Type: IO Command

Parameters: P1 – sound number to play : numeric variable or constant

Description: Start playing sound number P1 immediately.

Example:

```
IF SHOT?
THEN
SOUND(4)
ENDIF
```

If the current object has been shot then start sound number 4.

Flags:

Notes:

See Also: SYNCSND

Mnemonic: START **Shorthand:**

Type: Animator Command

Parameters:

Description: Marks the point in the animator code where the animator actually starts. This is the point that execution returns to when a RESTART is performed.

Example:

Flags:

Notes: Only valid in ANIMATORS.

See Also: RESTART

Mnemonic: STARTANIM(P1[,P2]) **Shorthand:**

Type: Animator Command

Parameters: P1 – animator number : numeric variable or constant

P2 – area number : numeric variable or constant

Description: Start the ANIMATOR condition P1 that is in area P2. If P2 is omitted the the ANIMATOR is assumed to be in the current area.

Example:

```
STARTANIM(3)
```

ANIMATOR number 3 will be started.

Flags:

Notes: If an ANIMATOR is started that is not in the current area, it will be marked as ready, but will not be activated until the player is actually in the area it belongs to.

See Also: STOPANIM, RESTARTANIM

Mnemonic: STARTANIMBRUSH(P1,P2,P3) **Shorthand:**

Type: Graphics Command

Parameters: P1 – brush number : numeric variable or constant

P2 – X coord : numeric variable or constant

P3 – Y coord : numeric variable or constant

Description: Start the anim brush number P1 at the screen location P2,P3.

Example: STARTANIMBRUSH(1,10,15)

Anim brush number 1 will be started at location 10,15

Flags:

Notes:

See Also: STOPANIMBRUSH

Mnemonic: STOPANIM(P1[,P2]) **Shorthand:**

Type: Animator Command

Parameters: P1 – ANIMATOR number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Stop the animator condition P1, in area P2, from executing. If P2 is omitted then the current area is assumed.

Example: STOPANIM(4)

Animator number 4 will be stopped.

Flags:

Notes:

See Also: STARTANIM, RESTARTANIM

Mnemonic: STOPANIMBRUSH(P1) **Shorthand:**

Type: Graphics Command

Parameters: P1 – Brush number : numeric variable or constant

Description: Stop the anim brush P1.

Example: STOPANIMBRUSH(1)

Flags:

Notes:

See Also: STARTANIMBRUSH

Mnemonic: STREQ?(P1,P2) **Shorthand:**

Type: Condition Statement

Parameters: P1 – string variable or constant

P2 – string variable

Description: Check if the two strings, P1 and P2, are identical.

Example:

```
IF STREQ?("END",S6)
THEN
PSTRING("Game Over",0,100)
ENDIF
```

If S6 is equal to "END" then "Game Over" is displayed on the screen.

Flags: TRUE if strings match

Notes:

See Also: SETSTR

Mnemonic: SUBVAR(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – numeric variable or constant

P2 – numeric variable

Description: Subtract P1 from P2 and put the results in P2.

Example:

```
SETVAR(10,V30)
SETVAR(15,V31)
SUBVAR(V30,V31)
```

V31 will now be equal to 5.

Flags: TRUE if result is zero

Notes:

See Also: ADDVAR, CLEARVAR, SETVAR

Mnemonic: SWAPSTR(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1,P2 – string variables

Description: Swap the strings P1 and P2.

Example: SETSTR("HELO",S1)

SETSTR("WORLD",S2)

SEWAPSTR(S1,S2)

S1 will now equal "WORLD" and S2 will now equal "HELLO"

Flags:

Notes:

See Also: SETSTR

Mnemonic: SWAPVAR(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1,P2 – numeric variables

Description: Swap the values of P1 and P2

Example: SETVAR(10,V77)

SETVAR(-17,V62)

SWAPVAR(V77,V62)

V62 will now equal 10 and V77 will now equal -17.

Flags:

Notes:

See Also: SETVAR

Mnemonic: SYNCSND(P1) **Shorthand:**

Type: IO Command

Parameters: P1 – sound number

Description: Synchronise sound number P1 with the start of the next frame.

Example: SYNCSND(10)

Sound number 10 will be start on the next frame.

Flags:

Notes:

See Also: SOUND

Mnemonic: TAN?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – object number : numeric variable or constant

P2 – area number : numeric variable or constant

Description: Check to see if object P1, in area P2, is tangible. If P2 is omitted then the current area is assumed.

Example:

```
IF TAN?(7)
THEN
MAKEWIRE(7)
ENDIF
```

If object 7 is currently tangible then it will be made wireframe.

Flags: TRUE if object is tangible

Notes:

See Also: INTAN?, MAKEINTAN, MAKETAN

Mnemonic: TESTMODE? **Shorthand:**

Type: Condition Statement

Parameters:

Description: Check if FreeScape system is currently running in TEST mode, in the editor.

Example:

```
IF TESTMODE?
THEN
PSTRING("Test Mode",0,0)
ENDIF
```

If running in test mode then the string "Test Mode" will be printed on the screen.

Flags: TRUE if running in test mode.

Notes: This can be a useful instruction for debugging purposes, as any debugging displays generated inside a TESTMODE condition will not be displayed when a game is made using MAKE.

See Also:

Mnemonic: TEXTCOL(P1,P2) **Shorthand:**

Type: Graphics Command

Parameters: P1 – foreground colour : numeric variable or constant

P2 – background colour : numeric variable or constant

Description: Set the foreground colour to P1 and the background colour to P2 for text printing.

Example: TEXTCOL(5,3)

PSRINT("Hello World",0,0)

"Hello World" will be printed in colour 5 on a background of colour 0 at screen location (0,0)

Flags:

Notes:

See Also: TEXTFONT

Mnemonic: TEXTFONT(P1) **Shorthand:**

Type: Graphics Command

Parameters: P1 – New font number : numeric variable or constant

Description: Set the font to use for printing to P1.

Example:

Flags:

Notes:

See Also: TEXTCOL

Mnemonic: THEN **Shorthand:**

Type: Condition Statement

Parameters: Marks the beginning of a block of command to be executed if the preceding condition(s) were TRUE, otherwise execution passes to the relevant ELSE if it exists otherwise to the relevant ENDIF.

Description:

Example:

Flags:

Notes:

See Also: IF, ELSE, ENDIF

Mnemonic: TIME(P1,P2,P3) **Shorthand:**

Type: IO Command

Parameters: P1 – Hours numeric variable

P2 – Minutes numeric variable

P3 – Seconds numeric variable

Description: Get the time from the system and place the hours in P1, minutes in P2, and the seconds in P3.

Example:

Flags:

Notes:

See Also: DATE

Mnemonic: TIMER? **Shorthand:**

Type: Condition Statement

Parameters:

Description: Checks the state of the condition timer.

Example:

Flags: TRUE if the timer value set in the Timer Delay section of the DEFAULTS menu has been reached.

Notes: Does not operate correctly in object conditions.

See Also:

Mnemonic: TOASCII(P1,P2,P3) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – numeric variable

P2 – conversion length : numeric variable or constant

P3 – destn string

Description: Convert P1 to an ASCII string and place the LAST P2 characters in to the destination string,

Example: SETVAR(123456,V30)

TOASCII(V30,3,S1)

S1 will be set to "456"

Flags:

Notes:

See Also: FROMASCII

Mnemonic: TOGBIT(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – bit number : numeric variable or constant

P2 – numeric variable

Description: Toggle bit number P1 in variable P2.

Example: SETVAR(5,V30)

TOGBIT(2,V30)

V30 will now equal 1

Flags: TRUE if result is zero

Notes: WARNING: Be careful about toggling the sign bit (bit 31) as this may lead to erratic results.

See Also: SETBIT, CLEARBIT, BITCLEAR?, BITSET?

Mnemonic: TOGVIS(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – object number : numeric variable or constant

P2 – area number : numeric variable or constant

Description: Toggle the visibility of object number P1, in area P2 (ie if its visible, make it invisible and vice-versa). If P2 is omitted then the object is assumed to be in the current area.

Example:

Flags:

Notes:

See Also: INVIS, INVIS?, VIS, VIS?

Mnemonic: TOLOWER(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – string variable

Description: Force all upper case character to be lower case in the

string

Example: SETVAR("HELLO World",S7)
TOLOWER(S7)
S7 will now contain "hello world"

Flags:

Notes:

See Also: TOUPPER

Mnemonic: TOUCH(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number to touch : numeric variable or constant
P2 – Area number : numeric variable or constant

Description: Force object P1 in area P2 to be flagged as touched. If P2 is omitted then the current area is assumed.

Example:

Flags:

Notes:

See Also: TOUCHED?

Mnemonic: TOUCHED?(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant
P2 – Area number : numeric variable or constant

Description: Check if the object P1, in area P2, has been touched. If P2 is omitted then the current area is assumed.

Example:

Flags: TRUE if object has been touched

Notes:

See Also: TOUCH

Mnemonic: TOUPPER(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – string variable

Description: Force all characters in P1 to be upper case.

Example: SETVAR("HELLO World",S10)

TOUPPER(S10)

S10 will now contain "HELLO WORLD"

Flags:

Notes:

See Also: TOLOWER

Mnemonic: TRIGANIM(P1[,P2]) **Shorthand:**

Type: Animator Command

Parameters: P1 – ANIMATOR number : numeric variable or condition

P2 – Area number : numeric variable or condition

Description: Mark ANIMATOR condition P1, in area P2, as triggered. If P2 is omitted then the ANIMATOR is assumed to be in the current area.

Example: IF SHOT?

THEN

TRIGANIM(5)

ENDIF

If the current object has been shot then trigger animator 5

Flags:

Notes:

See Also: WAITTRIG

Mnemonic: TRIGGENERAL(P1) **Shorthand:**

Type: Execution Modifier

Parameters: P1 – General condition number : numeric variable or constant

Description: Trigger the general condition P1

Example: TRIGGENERAL(4)

General condition number 4 will be triggered.

Flags:

Notes:

See Also: WAITTRIG

Mnemonic: TRIGLOCAL(P1) **Shorthand:**

Type: Execution Modifier

Parameters: P1 – Local condition number : numeric variable or constant

Description: Trigger the local (area) condition P1.

Example: TRIGLOCAL(7)

Local condition number 7 will be triggered

Flags:

Notes:

See Also:

Mnemonic: TRIGOBJECT(P1[,P2]) **Shorthand:**

Type: Execution Modifier

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Trigger the object condition associated with object number P1, in area P2. If P2 is not specified then the current area is assumed.

Example: TRIGOBJECT(4)

The conditions for object number 4 will be triggered

Flags:

Notes:

See Also: WAITTRIG

Mnemonic: UMOVE(P1,P2,P3) **Shorthand:**

Type: System Modifier

Parameters: P1 – Delta X value : numeric variable or constant

P2 – Delta Y value : numeric variable or constant

P3 – Delta Z value : numeric variable or constant

Description: Move the player by the relative amounts in each axis, with collision detection.

Example:

```
IF SHOT?
THEN
UMOVE(0,100,0)
ENDIF
```

If the current object has been shot then move the player 100 units in the Y axis.

Flags: TRUE if no collision occurred.

Notes:

See Also: UMOVETO

Mnemonic: UMOVETO(P1,P2,P3) **Shorthand:**

Type: System Modifier

Parameters: P1 – X position : numeric variable or constant

P2 – Y position: numeric variable or constant

P3 – Z position: numeric variable or constant

Description: Move the player to the absolute position in each axis, with collision detection.

Example:

```
IF SHOT?
THEN
UMOVETO(0,0,0)
ENDIF
```

If the current object has been shot then move the player 1to the location 0,0,0

Flags: TRUE if no collision occurred.

Notes:

See Also: UMOVE

Mnemonic: UNDEFARRAY(P1) **Shorthand:**

Type: Variable Modifier

Parameters: P1 – array variable

Description: Remove the definition for array P1 from the system and return its memory to the free memory pool

Example:

Flags:

Notes:

See Also: DEFARRAY, CLEARARRAY

Mnemonic: UNDESTROY(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Mark object P1, in area P2, as not destroyed. If P2 is not specified then the object is assumed to be in the current area.

Example:

Flags:

Notes:

See Also: DESTROY, DESTROYED?

Mnemonic: UNLOCK(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Set object number P1, in area P2, to be unlocked. If P2 is not specified then then object is assumed to be in the current area.

Example:

Flags:

Notes:

See Also: LOCK, LOCKED?, UNLOCKED?

Mnemonic: UNLOCKED?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Check if object P1, in area P2, is currently unlocked. If P2 is not specified then the current area is assumed.

Example:

```
IF UNLOCKED?(2)
THEN
LOCK(2)
ENDIF
```

If object 2 is currently unlocked then mark it as locked.

Flags: TRUE if object is unlocked
Notes:
See Also: LOCK, LOCKED?, UNLOCK

Mnemonic: UPDATEI(P1) **Shorthand:**
Type: Graphics Command
Parameters: P1 – Instrument number : numeric variable or constant
Description: For instrument P1 to be updated at the next frame.
Example:
Flags:
Notes:
See Also:

Mnemonic: VAREQ?(P1,P2) **Shorthand:** VAR=?
Type: Condition Statement
Parameters: P1 – numeric variable or constant
P2 – numeric variable
Description: Check if P1 is equal to P2.
Example:
Flags: TRUE if P1=P2
Notes:
See Also: VARGE?, VARGT?, VARLE?, VARLT?

Mnemonic: VARGE?(P1,P2) **Shorthand:** VAR>=?
Type: Condition Statement
Parameters: P1 – numeric variable or constant
P2 – numeric variable or constant
Description: Check if P1 is greater than or equal to P2.
Example:
Flags: TRUE if P1>=P2
Notes:
See Also: VAREQ?, VARGT?, VARLE?, VARLT?

Mnemonic: VARGT?(P1,P2) **Shorthand:** VAR>?
Type: Condition Statement
Parameters: P1 – numeric variable or constant
P2 – numeric variable or constant
Description: Check if P1 is greater than P2.
Example:
Flags: TRUE if P1>P2
Notes:
See Also: VAREQ?, VARGE?, VARLE?, VARLT?

Mnemonic: VARLE?(P1,P2) **Shorthand:** VAR<=?
Type: Condition Statement
Parameters: P1 – numeric variable or constant
P2 – numeric variable or constant
Description: Check if P1 is less than or equal to P2.
Example:
Flags: TRUE if P1<=P2
Notes:
See Also: VAREQ?, VARGE?, VARGT?, VARLT?

Mnemonic: VARLT?(P1,P2) **Shorthand:** VAR<?
Type: Condition Statement
Parameters: P1 – numeric variable or constant
P2 – numeric variable or constant
Description: Check if P1 is less than P2.
Example:
Flags: TRUE if P1<P2
Notes:
See Also: VAREQ?, VARGE?, VARGT?, VARLE?

Mnemonic: VIEWWINDOW(P1,P2,P3,P4) **Shorthand:**
Type: Graphics Command

Parameters: P1 – X left : numeric variable or constant

P2 – Y top: numeric variable or constant

P3 – X right : numeric variable or constant

P4 – Y bottom: numeric variable or constant

Description: Set the view window to the specified coords.

Example: VIEWWINDOW(0,0,319,199)

Flags:

Notes:

See Also:

Mnemonic: VIS(P1[,P2]) **Shorthand:**

Type: Object Command

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Set the attributes for object P1, in area P2, so that it visible. If P2 is not specified then the object is assumed to be in the current area.

Example: IF INVIS?(4)

THEN

VIS(4)

ENDIF

Object 4 will be made visible

Flags:

Notes:

See Also: INVIS, TOGVIS, INVIS?, VIS?

Mnemonic: VIS?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Check the attributes for object P1, in area P2, to see if it visible. If P2 is not specified then the object is assumed to be in the current area.

Example: IF VIS?(4)

THEN
INVIS(4)
ENDIF
Object 4 will be made invisible

Flags: TRUE if object is visible
Notes:
See Also: INVIS, INVIS?, TOGVIS, VIS

Mnemonic: VIST(P1) **Shorthand:**
Type: System Modifier
Parameters: P1 – Area number : numeric variable or constant
Description: Force area number P1 to be flagged as visited
Example:
Flags:
Notes:
See Also: VISITED?

Mnemonic: VISITED?(P1) **Shorthand:**
Type: Condition Statement
Parameters: P1 – Area number : numeric variable or constant
Description: Check to see if area number P1 has been visited.
Example:
Flags: TRUE if area has been visited
Notes:
See Also: VISIT

Mnemonic: WAIT **Shorthand:**
Type: Execution Modifier
Parameters:
Description: Stop execution until the next Freescape frame an return execution to the Freescape kernel. When the next frame occurs, execution will continue from the next instruction automatically.
Example:

Flags:

Notes:

See Also:

Mnemonic: WAITTRIG **Shorthand:**

Type: Execution Modifier

Parameters:

Description: Stop execution of this condition and return control to the Freescape kernel. Execution will continue from the next instruction when the relevant trigger command is issued for this condition.

Example:

Flags:

Notes:

See Also: TRIGANIM, TRIGGENERAL, TRIGOBJECT, TRIGLOCAL

Mnemonic: WIRE?(P1[,P2]) **Shorthand:**

Type: Condition Statement

Parameters: P1 – Object number : numeric variable or constant

P2 – Area number : numeric variable or constant

Description: Check the attributes for object P1, in area P2, to see if it is wireframe. If P2 is not specified then the object is assumed to be in the current area.

Example:

```
IF WIRE?(4)
THEN
MAKESOLID(4)
ENDIF
```

Object 4 is made solid.

Flags: TRUE if object is wireframe.

Notes:

See Also: SOLID?, MAKESOLID, MAKEWIRE

Mnemonic: XORVAR(P1,P2) **Shorthand:**

Type: Variable Modifier

Parameters:	P1 – numeric variable or constant
	P2 – numeric variable
Description:	Bitwise exclusive or P1 with P2 and put the result in P2.
Example:	SETVAR($AA,V30)
	XORVAR($FF,V30)
	V30 will now have the value $55
Flags:	TRUE if the result is zero
Notes:	
See Also:	ANDVAR, ORVAR, NEGVAR, NOTVAR

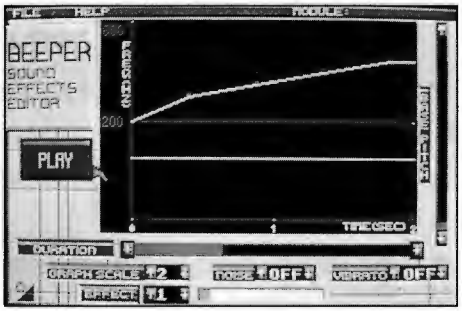
13: THE SOUND EDITORS

OK, so you've created a Freescape world that is a veritable orgy of colour, shape and movement. But still its not quite right, there still seems to be something missing : yes, that's right, we need to make a veritable orgy of colour, shape, movement and SOUND! Included with each version of the program is at least one type of sound editor.

PC

Included with the PC version are not one, not two, but three different sound editors. These are for the Ad Lib sound card and compatibles, the Roland LAPC-1 sound card and the good old built in beeper. To run the sound editor, exit 3DEDIT and type 3DSOUND at the DOS prompt. You will then be presented with a menu with the three sound cards – the ones you actually have installed will be marked with an asterisk.

Beeper



If you select Beeper then you will see a screen like fig. 13.1. At the top of the screen is a menu bar which allows you to load and save individual sound effects, load and save modules (collections of sound effects) for 3D Construction Kit 2.0, to get help and to quit.

Figure 13.1

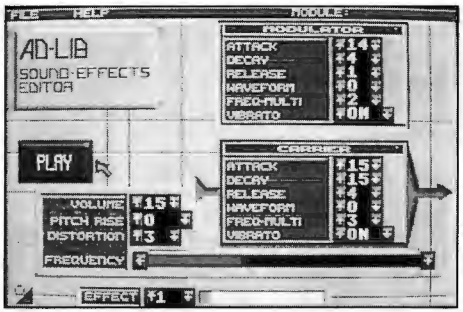
At the top of the main screen you will see a PLAY button – use this to play your sound effect. Next to that is a big window which allows you to edit the sound effect waveform. To the right of this is a bar which allows you to change the base pitch (frequency) that the sound effect will be played at. Underneath the edit window is another bar which allows you to change the duration of the sound effect. Underneath that is the graph scale setting which changes the amount of space available to show the waveform. Next to that are two buttons one which turns noise on and off and one which turns vibrato on and off.

Finally, at the bottom of the screen is a box that allows you to change the name of the current effect.

A beeper sound effect is made up by first deciding what the base pitch will be. Within the edit window is a waveform with four small squares on it. You can click

on these squares and drag them around the screen to define what pitch, above or below the base pitch, the sound effect will have at that particular time.

Ad-Lib



If you select Ad-Lib then you will see a screen like fig. 13.2. At the top of the screen is a menu bar which allows you to load and save individual sound effects, load and save modules (collections of sound effects) for 3D Construction Kit 2.0, to get help and to quit.

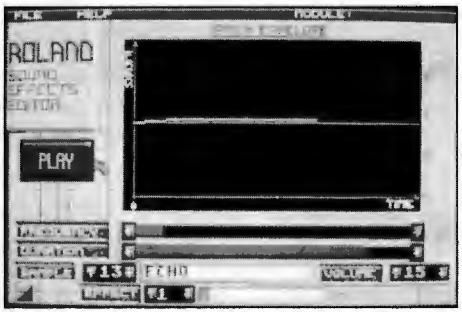
Figure 13.2

At the top of the main screen you will see a PLAY button – use this to play your sound effect. Next to this are two windows which allows you to edit the parameters for both the modulator and carrier of the FM synthesis waveform. To the left of these are buttons which allow you to change the volume, change in pitch with time and the level of distortion applied to the waveform. There is also a bar which allows you to change the base pitch (frequency) that the sound effect will be played at.

Finally, at the bottom of the screen is a box that allows you to change the name of the current effect.

See your Ad-Lib documentation for more information.

Roland



If you select Roland then you will see a screen like fig. 13.3. At the top of the screen is a menu bar which allows you to load and save individual sound effects, load and save modules (collections of sound effects) for 3D Construction Kit 2.0, to get help and to quit.

Figure 13.3

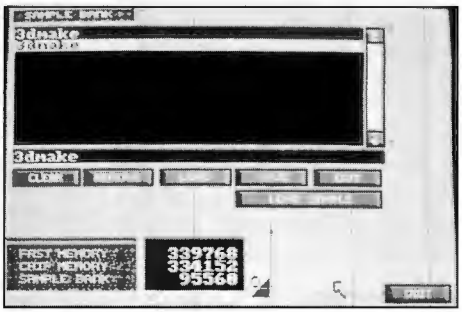
At the top of the main screen you will see a PLAY button – use this to play your sound effect. Next to that is a big window which allows you to edit the way the waveform pitch changes with time. Underneath this is a bar which allows you to change the base pitch (frequency) that the sound effect will be played at, and a bar to change the duration of the sound effect. Underneath that is the SAMPLE box which changes the sample which is used for the sound effect and a box to change the overall sound effect volume.

Finally, at the bottom of the screen is a box that allows you to change the name of the current effect.

Within the edit window is a line with three small squares on it. You can click on these squares and drag them around the screen to define what pitch, above or below the base pitch, the sound effect will have at that particular time.

See your Roland documentation for more information.

Amiga



To load the Amiga sound editor quit the 3D editor and either type 3DSOUND from CLI or double click on its icon from Workbench. You will be presented with a screen like fig. 13.4. At the top of the screen is the sample selector window. For the current module this lists all the samples built in to it. Below this window is a series of buttons :

Figure 13.4

CLEAR	Remove all samples from the sample bank.
REMOVE	Remove highlighted sample from sample bank.
LOAD	Load a new sample bank.
SAVE	Save the current sample bank out.
EDIT	Edit the highlighted sample

LOAD SAMPLE Load an individual sample in to the editor

QUIT Leave the sound editor

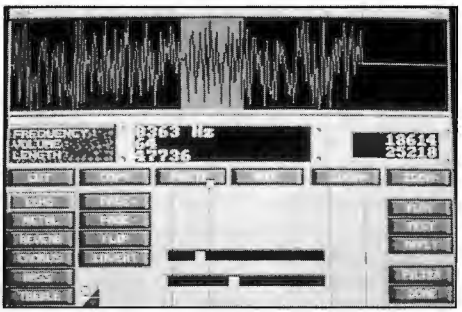


Figure 13.5

When you click on EDIT you will be presented with a screen like fig. 13.5. At the top of this screen is the sample view window. This gives a graphical representation of the sound waveform. Using the mouse you can mark a section of the waveform to be manipulated. Underneath this are a series of waveform edit buttons (these only work on a marked range).

CUT Cut the marked range and put in the paste buffer

COPY Make a copy of the marked range and put it in the paste buffer

PASTE Insert the paste buffer at the beginning of the marked area

MIX Mix the contents of the paste buffer with the currently marked area

ZOOM+ Zoom in on range

ZOOM- Zoom out from range

To the left and below these buttons are the effects buttons (again these only work on a marked range). Next to the buttons are two sliders which control level and duration for each of the effects.

ECHO Echo marked range

METAL Apply a "Heavy Metal" distortion

REVERB	Apply reverb to range
CHORUS	Apply a chorus effect to range
BASS	Boost bass frequencies
TREBLE	Boost treble frequencies
FADE+	Fade sample volume up from 0
FADE-	Fade sample volume down to 0
FLIP	Reverse range
INVERT	Invert range

NB None of these effects are actually applied until the APPLY button is used.

To the right of the screen are the playback buttons :

PLAY	Play the sample
TEST	Test the sample with the effect requested
APPLY	Apply the current effect to the sample
FILTER	Turn the internal hardware filter on and off
DONE	Return to sample editor

Atari ST

See documentation file on disk for details.

14: THE MAKE UTILITY

So, once you've created your symphony of vision and sound, what can you do with it? Well the answer is, using the Make utility, you can create a standalone executable file that you can distribute freely to amaze and confound your friends. Before you use Make, first make sure that you have saved you entire world out (using SAVE DATA from the FILE menu in 3DEDIT). At this point, if you are intending to save you executable to floppy make sure you have a formatted disk with enough space to save all you data, if you are saving to hard drive, make sure that you have enough free space. Type 3DMAKE to run the Make utility.

Once in Make, you will asked to give a seven character name which will be used to start your executable program running. You will then be asked to locate the drive and directory where all the data associated with your file is saved. Make will create a new sub-directory in your chosen directory which will have the name you entered earlier preceded by an underscore ("_"). This sub-directory will contain all the data, borders and sound files associated with your world will be stored. For this reason, any directories with a leading underscore will not be selectable, so you cannot create a new environment inside a previously created environment directory.

At this point Make will save all the relevant system files and then the run-time Freescape system ("Runner"). Next you will be asked to locate you data file (.3WD extension) that you saved your world in. Your data file will then be saved off and scanned to see if any borders are needed. If so these are then saved as well.

To execute your standalone environment, simply type the name you chose for your executable program.

APPENDIX 1

A1.1 Installation and Loading Instructions

PC

The PC version can only be run from hard drive. Insert disk 1 into the floppy drive and type INSTALL and follow the on-screen prompts. Included with your disk are a number of sample objects ("THE CLIP-ART CATALOGUE") and sound editors for the most popular sound boards on the PC, you may choose to not install these by answering No at the correct prompt. Once installed, you must run 3DSETUP before running the main program for the first time. This will allow you to choose the graphics mode, default sound board and type of expansion memory to use. To run the editor type 3DEDIT.

Amiga

The Amiga version can either be run from floppy or from hard drive. Before you do anything : MAKE A BACKUP. If running from floppy then simply double click on the 3DEDIT icon or type 3DEDIT from the cli prompt. To install to hard drive double click on the INSTALL icon and follow the on-screen prompts.

NB 3D Construction Kit 2.0 is packed full of features, and as a result it need a lot of memory to run. Certain versions of the Amiga (notably the 1Meg machines with Workbench 2 and hard drives) use a lot of memory for resident drivers etc. and this may cause problems. Before running the program ensure you have as few memory resident utilities and system files as possible, if you then run the program and the title screen does not appear, the try closing all windows (including the WorkBench window). Alternatively, we have provided a modified STARTUP-SEQUENCE for WB2 which allows you to remove a lot of the memory overhead.

Atari ST

The ST version can either be run from floppy or from hard drive. Before you do anything : MAKE A BACKUP. If running from floppy then simply double click on the 3DEDIT icon. To install to hard drive, copy all files and subdirectories from both disks into a sub-directory on your hard drive.

A1.2 Filename Extensions

.3WD	World data file
.3AD	Area data file
.3OD	Object data file
.3NA	Ad Lib sound file (PC only)
.3NB	Beeper sound file (PC only)
.3NR	Roland sound file (PC only)
.3SM	AMIGA/ST Sample file
.TXT	Condition text file
.IFF	Brush/Border file extension

A1.3 Loading Objects / Worlds from the Clip-Art Catalogue

First select the object or world that you want from the catalogue. Underneath the picture you will see a name. If this ends in .3OD then you need to select LOAD OBJECT else select LOAD DATA, both on the FILE menu. Using the file selector, find the clip art directory, which is a sub-directory of the drive where you are running 3D Construction Kit 2.0 from. Type in the filename and click on OK.

A1.4 Importing Objects from 3D Construction Kit 1.0

Due to the enhancements we have made to 3D Construction Kit 2.0 it was not possible to make areas and objects saved in 3D Construction Kit 1.0 format directly loadable into 3D Construction Kit 2.0. However, we have included the facility to load world (data) files, so if you load 3D Construction Kit 1.0 and save out your object or area in data format, you will then be able to load it into 3D Construction Kit 2.0.

CAVEAT: 3D Construction Kit 1.0 allowed objects to have a maximum coordinate value of 8192. This will cause problems if loaded into 3D Construction Kit 2.0. To get round this problem, edit your object in 3D Construction Kit 1.0 to make sure all coordinates are less than 8192.

APPENDIX 2 : Hints and Tips or Dr. Bozz`s FCL 3D Construction Kit 2.0 Clinic

I have an box object which I want to duplicate. It has a condition attached to it that makes it disappear when it is shot. Unfortunately, when I do duplicate it the condition on the new object does not work.

The problem is that you have used an absolute object number in your object condition :

```
IF SHOT?
THEN
        INVIS(2)
ENDIF
```

When you duplicate the object, the duplicate will have a different object number to the original and so the INVIS will not work. To get round the problem use the special system variable ME, which contains the object number of the current object.

```
SETVAR(ME,V511)
IF SHOT?
THEN
        INVIS(ME)
ENDIF
```

Now, no matter how many times you copy the object, it will always disappear!

I`ve built a house with an outside view. Unfortunately, I can`t see out of my windows.

Well try cleaning them then! (Sorry, my little joke). Sounds like a job for the good old FADEVAL. Try setting the window object fade value (follow OBJECT->ATTRIBUTES) to a value other than 0. For a different effect edit one of the user fade values to give a cross-hatch appearance.

I`m still not sure about the different types of invisibility.

OK. There are basically three types of invisibility. Firstly, there is colour invisibility. On the colour select panel, the first colour available (marked with an I) is transparent : any facet coloured with will have all the calculations associated with it performed, but will not be drawn. Note that an object that is completely coloured Invisible will still exist in the world and can still be collided with, shot etc. The second type of invisibility is a facet with a fade value of 9, this is like the colour invisible but the facet does actually have a colour, you just can`t see it (if you see what I mean). The final type of invisibility is an object that has its INV attribute set (OBJECT->ATTRIBUTES) : this is like setting all the facets of an object to Invisible colour at once and makes it invisible to collisions aswell.

How do I print out my conditions so I can amaze and entertain my friends?

Condition files are simply normal text files, so you can print them in the same way you normally print files.

When should I use SYNCSND and when should I use SOUND.

If you remember from the reference section SOUND generates sounds as soon as the SOUND command is executed. SYNCSND however will not start generating sounds until the next frame update. SOUND is useful, therefore, if you have to have immediate feedback on something the program does. For instance, if you have a button that can be clicked on that performs an action that may take a long time (for instance : loading a border from disk), you may want a confirming beep to let the user know that the program has noticed the button press. This beep will be no use if it takes a long time to come, so use SOUND to generate it immediately. SYNCSND is useful for those situations where you have a sound effect that could be generated more than once in one frame such as an explosion. For instance, you may have a number of objects that when shot generate an explosion sound. You don`t want the explosion to be continually re-triggered, so use SYNCSND.

APPENDIX 3 : KEYBOARD SHORTCUTS

Modes

ALT + t	Enter test mode
ALT + n	Create a new object
ALT + e	Edit object
ALT + s	Select object
ALT + i	Edit object attribute information
ALT + c	Colour object
SHIFT + SPACE	Shortcuts on/off

View Positions

ALT + v	Select new vehicle
ALT + F1	Change to WALK mode
ALT + F2	Change to FLY1 mode
ALT + F3	Change to FLY2 mode
ALT + F4	Change to camera 1 view
ALT + F5	Change to camera 2 view
ALT + F6	Change to camera 3 view
ALT + F7	Change to camera 4 view
ALT + F8	Change to camera 5 view
ALT + F9	Change to camera 6 view

Info

ALT + w	Set status window to world section
ALT + o	Set status window to object section
ALT + a	Set status window to area section
ALT + m	Set status window to memory section
ALT + h	Toggle highlight on/off
ALT + x	Toggle exclude on/off
ALT + q	Quit

FCL Debugger

RETURN	Execute command
SPACE	Skip command
ESC	Exit debugger
Up cursor	Scroll window up
Down cursor	Scroll window down
F1	Select new variable 1
F2	Select new variable 2
F3	Select new variable 3
F4	Select new string
w	Toggle Wait flag
t	Toggle waitTrig flag

(IBM version only)

PGUP	Move PC up
PGDOWN	Move PC down
HOME	Move to start of program
END	Move to end of program

The next step is...

SUPERSCAPE®

Interactive Virtual World Creation and Visualisation

Specifications include:-

☆	**WORLD SIZE**	4.39 Billion units cubed
☆	**SCREEN RESOLUTION**	Real time switchable up to 1280 x 1024
☆	**REAL TIME LIGHTING**	Multiple static or moving light sources
☆	**OBJECT & VIEW ROTATIONS**	0.05 degree resolution
☆	**CONTROL LANGUAGE**	SCL over 420 commands
☆	**TEXTURE MAPPING**	In VRT3 due for release Q1 1993
☆	**DYNAMICS**	Including friction, restitution, effects under gravity, rotational and translational velocities

PLUS

Spin and extrude shape creation, DXF import, open file format, networked virtual worlds, movement paths, direct object control, tweening of 3D Animations, multiple undo, file security, plus many other features.

Product Range includes:-

★ **SUPERSCAPE VRT.** Complete Interactive World Creation and editing system. Includes Superscape Visualiser, Shape Editor, World Editor, VR Clip Art and Sample Virtual Worlds.

★ **SUPERSCAPE NETWORK KIT.** Optional extension to SUPERSCAPE VRT to allow creation of Networked Virtual Worlds.

★ **SUPERSCAPE DEVELOPERS KIT.** Optional toolkit for programmers only wishing to interface SUPERSCAPE Virtual Worlds with 3rd Party applications or devices.

★ **SUPERSCAPE VISUALISER.** For use with stand alone run-time applications requiring Interactive Visualisation of SUPERSCAPE Virtual Worlds.

For a FREE Information Pack on the complete range of SUPERSCAPE products and services, please contact:-

DIMENSION
INTERNATIONAL

Zephyr One, Calleva Park, Aldermaston, Berkshire, England, RG7 4QZ
Tel: +44 (0)734 810077 Fax: +44 (0)734 816940

WARNING

It is a criminal offence to sell, hire, offer or expose for sale, or hire or otherwise distribute infringing (illegal) copies of this computer program and persons found doing so will be prosecuted.

Any information of piracy should be passed to The Federation Against Software Theft. 0628 660377.

PUBLISHED BY DOMARK SOFTWARE LTD.,
FERRY HOUSE, 51-57 LACY ROAD, PUTNEY, LONDON SW15 1PR